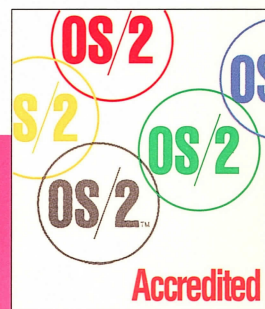


Instant OS/2! Porting C Applications to OS/2

- Migrate to 32-bit full-screen OS/2 with ease
- Covers OS/2 2.1 and IBM and Borland's C++ for OS/2 compilers
- Includes VIO, KBD, and MOU calls
- Disk includes all source code



Ken Dorfman



Instant OS/2[®]! Porting C Applications to OS/2

Instant OS/2[®]! Porting C Applications to OS/2

Len Dorfman

Windcrest[®]/McGraw-Hill

New York San Francisco Washington, D.C. Auckland Bogotá
Caracas Lisbon London Madrid Mexico City Milan
Montreal New Delhi San Juan Singapore
Sydney Tokyo Toronto

OS/2 Accredited logo is a trademark of IBM Corp. and is used by McGraw-Hill, Inc. under license. *Instant OS/2®!: Porting C Applications to OS/2* is independently published by McGraw-Hill, Inc. IBM Corp. is not responsible in any way for the contents of this publication.

McGraw-Hill, Inc. is an accredited member of the IBM Independent Vendor League.

FIRST EDITION
FIRST PRINTING

©1994 by **Windcrest Books**.

Published by Windcrest Books, an imprint of McGraw-Hill, Inc.

The name "Windcrest" is a registered trademark of McGraw-Hill, Inc.

Printed in the United States of America. All rights reserved. The publisher takes no responsibility for the use of any of the materials or methods described in this book, nor for the products thereof.

Library of Congress Cataloging-in-Publication Data

Dorfman, Len.

Instant OS/2! : porting C applications to OS/2 / by Len Dorfman.

p. cm.

Includes index.

ISBN 0-8306-4522-5 (paper)

1. Operating systems (Computers) 2. OS/2 (Computer file) 3. C
(Computer program language) I. Title.

QA76.76.063D67 1993

005.4'469—dc20

93-10538

CIP

Acquisitions Editor: Jennifer DiGiovanna

Editorial Team: Robert E. Ostrander, Executive Editor

David M. McCandless, Book Editor

Director of Production: Katherine G. Brown

Book Design: Jaclyn J. Boone

Marble paper background courtesy of Douglas M. Parks, Blue Ridge Summit, Pa.
Cover design and marble paper by Margaret Karczewski, Vienna, Va.

Contents

<i>Preface</i>	<i>xi</i>
<i>Introduction</i>	<i>xiii</i>

PART ONE:

DEMONSTRATION PROGRAMS AND CHARACTER MODE API

1	Developing a uniform full screen OS/2 & DOS character mode API	3
	Function prefix list	4
	The uniform OS/2 full screen and DOS character mode API	4
	Building your libraries and demonstration programs	18
	Summary	22
2	Keyboard management demonstration programs	23
	Keyboard function descriptions	24
	Function kb__read()	24
	Function kb__scan()	24
	Function kb__char()	24

Function kb__status()	24
Function kb__edit(...)	25
Stopping program execution & reading a 16-bit key code	25
Stopping program execution & reading an 8-bit character code	26
Stopping program execution & reading an 8-bit scan code	28
Not stopping program & retrieving a 16-bit key code	29
Retrieving a character string from the keyboard	31
Summary	33

3 Text cursor management demonstration programs 35

Cursor function descriptions	36
Function cu__get__loc(...)	36
Function cu__get__shape()	36
Function cu__set__shape(...)	36
Function cu__move(...)	36
Function cu__relative__move(...)	36
Function: cu__remove()	37
Function: cu__display()	37
Function cu__save__size()	37
Function cu__rest__size()	37
Function cu__set__size(...)	37
Function cu__save__loc()	37
Function cu__rest__loc()	38
Manipulating the cursor location, size, and visibility	38
Summary	41

4 Screen management demonstration programs 43

Screen function descriptions	44
Function scrn__init()	44
Function scrn__save()	45
Function scrn__restore()	45
Function scrn__clear()	45
Function scrn__write(...)	45
Function scrn__char(...)	45
Function scrn__chr(...)	46
Function scrn__repeat__char(...)	46
Function scrn__attr(...)	46
Function scrn__change__attr(...)	47
Function scrn__read__char(...)	47
Make functions description	47
Function mk__attr(...)	47
Function mk__attr__intense(...)	47
Function mk__attr__inverse(...)	48
Function mk__attr__blink(...)	48

Function mk_attr_intense_blink(...)	48
Function mk_token(...)	48
Function mk_char_attr(...)	49
Demonstration of useful screen management routines	49
Introduction to menu bar / drop down window creation	52
Summary	68
5 Mouse management demonstration programs	69
Mouse function descriptions	69
Function ms_init()	69
Function ms_on()	70
Function ms_off()	70
Function ms_status(...)	70
Function ms_map_display(...)	70
Map your full screen OS/2 or DOS display using the mouse	71
Map the user interface presented in PROG4-2	73
PROG4-2's interface becomes keyboard and mouse driven	78
Summary	104
6 Character-based window management demonstration programs	105
Window function descriptions	107
Function wind_init(...)	107
Function wind_kb_edit(...)	108
Function wind_display(...)	108
Function wind_remove(...)	108
Function wind_attr(...)	109
Function wind_write(...)	109
Function wind_char(...)	110
Function wind_repeat_char(...)	110
Function wind_destroy(...)	111
Function wind_read_char(...)	111
Function wind_cu_move(...)	111
Mapping a dialog box	111
An about... dialog box demo program	115
An alphanumeric data field entry dialog box demo program	119
A multiple item check dialog box demo program	129
A string list dialog box	138
A file name and file size dialog box	141
A Lotus grid style menu demonstration program	143
A commercial quality setup program	155
Summary	217

7	Printer management demonstration programs	219
	Printer function descriptions	219
	Function print__open(...)	219
	Function print__close(...)	220
	Function print__newline(...)	220
	Function print__cr(...)	220
	Function print__string(...)	220
	Function print__char(...)	221
	Function print__set__column(...)	221
	A command line file print utility	221
	PROG7-1	221
	File print utility that puts it all together	228
	Summary	277

PART TWO

OS/2 FULL SCREEN CHARACTER MODE LIBRARY

8	OS/2 full screen keyboard management functions	281
9	OS/2 full screen cursor management functions	293
10	OS/2 full screen management functions	299
11	OS/2 mouse management functions	319
12	OS/2 full screen window management functions	323
13	OS/2 printer management functions	331

PART THREE

DOS CHARACTER MODE LIBRARY

14	DOS keyboard management functions	337
15	DOS cursor management functions	349
16	DOS screen management functions	355
17	DOS mouse management functions	375

18	DOS window management functions	379
19	DOS printer management functions	387
	Epilogue	391
	Index	393

Preface

I feel obligated to let you know that I believe the arrival of OS/2 2.1 to be one of the most important events to have occurred in the history of the 80386 (and up) chip world. The experience of using a very powerful full-fledged 32-bit multitasking operating system on my 386 and 486 systems has proven mind-boggling.

In retrospect, it seemed as if my 386 computer had always been functioning in a daze when powered under DOS or DOS overlays such as Windows. The daze became apparent only after I had installed OS/2 on my 386 computer. After I installed OS/2, my 386 became a more robust computer. More robust without a hardware change. Simply amazing! OS/2 turned my 486 local bus computer into a monster-class workhorse.

It didn't take me long to become an OS/2 believer. My coding and writing productivity improved beyond words within hours after installing OS/2 2.0 on my 33 Mz. 386 class machine. Instead of spending time waiting for my computer, I now spend time thinking about what else can I get my computer to do.

Know that OS/2 is not some software promise of being able to free up the raw power of the 386 and 486 microprocessors some time in the future. It's here now.

Introduction

There are far more character mode based programs running under DOS than programs running in either the Windows or OS/2 Presentation Manager environments. As much as industry leaders sing the praises of protected mode multitasking and graphical user interface based operating systems, DOS is still king in the real world. Simple as that.

I'll always be running Borland's Brief, my beloved text editor, in the 80 x 25 character mode. I have a far more difficult time reading graphically drawn fonts than character mode generated fonts. There will always be a significant gaggle of users who feel that text editors, word processors, spread sheets, etc., are easier on their eyes and head in character mode than in graphics mode.

Now that I've argued that there will always be a place for character mode programs on hard disks, then let's move on to why OS/2 Full Screen is a more robust environment than standard DOS. The answer lies in the realm of memory management.

DOS programmers know too well the machinations they have to go through when needing gobs of memory. There's EMS. There's XMS. There are virtual memory managers. What a mess.

OS/2 programmers don't fret one bit about memory. Why? OS/2 C programmers can allocate megabytes upon megabytes of memory with ease because OS/2 can run protected mode flat memory model programs. That means a program's data and code can be as large as four gigabytes. Case closed.

If you're writing a database management program and need one megabyte of memory for, say, a sort operation, simply allocate a megabyte via `malloc(...)`. Flat memory model management is simple and fast.

In short, migrating your existing DOS program so it can run in an OS/2 Full Screen character mode session means that:

- Users can stay with an interface they already feel comfortable with.
- Your program will become far more robust and agile in its memory management usage.
- Your program will execute memory intensive operations far faster in OS/2 full screen than under DOS.

Instant OS/2! should provide invaluable information for programmers wanting to migrate their real mode DOS character-based programs to OS/2's character mode 32-bit flat memory model.

Why full screen OS/2 as a starting point for migration

The software market changes so quickly it makes good sense to create market milestones as you convert existing applications to run under new generation 32-bit operating systems. These milestone upgrades should come often in order to keep your program current and steadily moving on the upgrade path.

OS/2's Presentation Manager API (Application Programmer's Interface) is of the same ilk as the Windows API or X-Windows's API, etc., and will take some time to master. Using the libraries presented in this book the differences between the DOS character mode API and OS/2 full screen API are quite small. You'll be able to get your 32-bit OS/2 full screen and DOS programs up and running in little time.

The book's organization

This book is divided into three distinct parts. Part One is composed of seven chapters. These chapters contain demonstration programs and function prototypes for full screen character mode OS/2 and DOS C libraries. These libraries contain same name and prototyped functions for the following categories:

- Keyboard management functions
- Cursor management functions
- Screen management functions
- Mouse management functions
- Character mode window functions
- Printer management functions

Part Two presents all the source code for the OS/2 full screen C library, and Part Three presents all the source code for the DOS character mode C library.

Part One

Demonstration programs and character mode API

Part One presents the source code to 21 demonstration programs. This source code may be compiled with either Borland's C++ or IBM's C/Set 2 Compiler. If you select the Borland compiler, you will generate real mode DOS programs. If you select the IBM compiler, you will generate 32-bit flat memory model full screen OS/2 programs.

Effort has made to minimize the conditional compilation statements present in the demonstration program source code. On rare occasions, calls will be made to dummy functions (functions containing no instructions) in order to maintain source code compatibility between DOS and OS/2.

I could have chosen to present conditional compilation statements in the source in lieu of the dummy functions but decided that the few bytes added to the executable code presented less of a penalty than the degradation of readability of the demonstration source code.

1

Developing a uniform full screen OS/2 & DOS character mode API

I remember the conversation well. My good friend (and deeply gifted programmer) Marc Neuberger called and barked the following message very strongly: "Install IBM's OS/2 Version 2.0 on your computer. Now!" Although Marc and I occasionally have different views on a variety of issues, I followed Marc's advice as if hypnotized. I'm glad I did.

I became hooked after my first day of using OS/2. Here's a capsule summary of what happened. I opened up a DOS window and began the somewhat long process of compressing and copying over 250 files from my hard disk to floppy. After I initiated this process I moved the mouse pointer over my 32-bit Presentation Manager telecommunications program and began dialing a BBS. The line was busy, so the telecommunications program receded to the background and I found myself staring at the computer.

I moved the mouse pointer over a Presentation Manager version of Reversi, clicked twice and began playing. After a few moves I fell into a feeling of awe. My disk compression and copy process was moving along at high speed. My telecommunications program was re-dialing the BBS. I was playing Reversi. I couldn't discern any degradation in application process performance. And this was on my 33 Mz. 386 no-name clone.

My reverie was broken when the BBS answered and the telecommunications program took over and my game of Reversi stayed waiting for my move.

Wow! I was hooked. OS/2 is a pure joy to use.

Function prefix list

I've decided to have standard function name prefixes for each function category in the OS/2 and DOS library presented in this book. Table 1-1 presents the function prefix list.

Table 1-1 Function prefix list.

Prefix	Category
cu_	Cursor
kb_	Keyboard
ms_	Mouse
mk_	Make
scrn_	Screen
wind_	Window
print_	Printer

The uniform OS/2 full screen and DOS character mode API

The purpose of a uniform API for OS/2 and DOS programmers is to be able to write a mouse and keyboard driven user interface, printer driver, or keyboard and mouse event handler, etc.—using one source file for both the OS/2 and DOS versions. This task has been accomplished by creating a uniform API for DOS and OS/2 programmers.

Figure 1-1 presents the source code listing for TPROTO.H. This file contains the function prototypes for the API presented in this book.

1-1 The source code listing to TPROTO.H.

```
////////////////////////////////////
//
// tproto.h
//
// Function prototype file for
// library functions
//
////////////////////////////////////
//
// include files here
//
#include "keyboard.h"
#include "ascii.h"
#include "tstruct.h"
//
////////////////////////////////////
//
// OS2 defines
//
```

```

#ifdef OS2_PROG
#ifdef BCOS2_COMP
#define itoa _itoa
#define ltoa _ltoa
#endif
#endif

////////////////////////////////////
//
// High level keyboard routines
//

int kb_edit(char *response,
            int row,
            int column,
            int dlen,
            int opt,
            UCHAR attr);
int kb_read(void);
int kb_status(void);
char kb_char(void);
UCHAR kb_scan(void);

////////////////////////////////////
// void kb_cap_on(void);
// void kb_cap_off(void);
// void kb_ins_in(void);
// void kb_ins_off(void);
// void kb_num_on(void);
// void kb_num_off(void);
// int gtkBflag(void);
// int gtkBflsh(int);
////////////////////////////////////

////////////////////////////////////
//
// High level cursor routines
//

void cu_get_loc(int *, int *);
int cu_get_shape(void);
void cu_move(int, int);
void cu_relative_move(int, int);
void cu_remove(void);
void cu_display(void);
void cu_save_size(void);
void cu_rest_size(void);
void cu_set_size(int, int);
void cu_save_loc(void);
void cu_set_shape(int);
void cu_rest_loc(void);

void putCR(void);
void putLF(void);
void putCRLF(void);

// rectangle routines

void vdBox(RECT *, int, unsigned char);
RECT *setRect(RECT *, int, int, int, int);

```

1-1 Continued.

```
void addRect(RECT *, RECT *);
void subRect(RECT *, RECT *);
void dupRect(RECT *, RECT *);
void dsyRect(RECT *);
void offRect(RECT *, int, int );
void boxRect(RECT *, int, UCHAR attr);
void clrRect(RECT *);
void fillRect(RECT *, int);
void saveRect(RECT *);
UINT sizeRect(RECT *);
void restRect(RECT *);
void scUp(RECT *, int, int);
void scDn(RECT *, int, int);

////////////////////////////////////
//
// High level print routines
//

int print_open(int num);
int print_close(int num);
int print_newline(int num);
int print_cr(int num);
int print_string(int num, char *);
int print_char(int, char);
int print_scrn(int);
int print_scrnFF(int);
int print_status(int);
void print_set_column(int, int);

////////////////////////////////////
//
// High level screen routines
//

void scrn_write(int row, int col, int len, char *str, UCHAR attr);
void scrn_init(void);
int scrn_read_char(unsigned short, unsigned short);
void scrn_save(void);
void scrn_restore(void);
void scrn_clear(void);
void scrn_attr(int, int, int, unsigned char);
void scrn_change_attr(unsigned char);
void scrn_char(int row, int col, char ch, UCHAR attr);
void scrn_repeat_char(int row, int col, int len, char ch, UCHAR attr);
void scrn_chr(int, int, int);

void ascup(int,int,int,int,int,int,int);
void putChr(char);
void putCRLF(void);
void putLF(void);
void putCR(void);
void putStr(char *);
int rdChar(void);
void vdHoriz(int, int, int, int);
int vdprompt(char *,int, int, int, int);
void vdStr(int, int, int, char *, char);
void vdVert(int, int, int, int);
```

```

void wrChar(char, int);

////////////////////////
//
// High level mouse routines
//

int ms_init(void);
void ms_on(void);
void ms_off(void);
int ms_status(int *x, int *y);
void ms_map_display(int row, int col, int key_val);

////////////////////////
//
// High level window routines
//

WIND *wind_init(WIND *W_PTR,
                int ulr,
                int ulc,
                int lrr,
                int lrc,
                UCHAR attr,
                int border,
                char *title );
int wind_kb_edit(WIND *W,
                 char *response,
                 int row,
                 int column,
                 int dlen,
                 int opt,
                 UCHAR attr);
void wind_display(WIND *);
void wind_remove(WIND *);
void wind_attr(WIND *, int, int, int, unsigned char);
void wind_write(WIND *, int, int, int, char *, UCHAR);
void wind_char(WIND *, int row, int col, char ch, UCHAR attr);
void wind_repeat_char(WIND *, int row, int col, int len, char ch, UCHAR
attr);
void wind_destroy(WIND *);
int wind_read_char(WIND *, int, int);
void wind_clear(void);
void wind_cu_move(WIND *, int, int);

////////////////////////
//
// Low level window routines
//

void setAttr(WIND *, unsigned char);
void setBord(WIND *, int);
void setTitle(WIND *, char *);
WIND *setWind(WIND *, int, int, int, int);
void strtWind(WIND *);
UINT sizeImg(WIND *);
void wrBox(WIND *);
void wrImg(WIND *);
void wrWind(WIND *);
void wvdHoriz(WIND *, int, int, int, int, int);
int wvdprmt(WIND *, char *, int, int, int, int);

```

1-1 Continued.

```
void wvdVert(WIND *, int,int,int,int);
void wvdScdn(WIND *, int);
void wvdScup(WIND *, int);
void wvdStr(WIND *, int, int, int, char *, char);
void rdImg(WIND *);
void rdWind(WIND *);

////////////////////////////////////
//
// Make Attributes and
// Tokens
//
////////////////////////////////////

UCHAR mk_attr(unsigned char,unsigned char,unsigned char,unsigned char);
UCHAR mk_attr_intense(unsigned char);
UCHAR mk_attr_intense_blink(unsigned char);
UCHAR mk_attr_inverse(unsigned char);
UCHAR mk_attr_blink(unsigned char);
void mk_char_attr(int token, char *ch, UCHAR *attr);
int mk_token(unsigned char, unsigned char);
```

1-1 Ends.

Figures 1-2 through 1-4 present other include files required by the OS/2 Full Screen and DOS programmer API. Figure 1-2 presents the source code listing to KEYBOARD.H. Figure 1-3 presents the source code listing to ASCII.H, which contains ASCII and miscellaneous defines. And FIG. 1-4 presents the source code listing to TSTRUCT.H, which contains structures used by the OS/2 Full Screen and DOS programmer API.

1-2 The source code listing to KEYBOARD.H.

```
////////////////////////////////////
//
// keyboard.h
//
// 16-bit Key Codes
//
////////////////////////////////////

#define INSERT      0x5200
#define DELETE     0x5300
#define SPACE      0x3920
#define ESC        0x011b
#define ESCAPE     0x011b
#define PGDN       0x5100
#define PGUP       0x4900
#define PERIOD     0x342e
#define TAB        0x0f09
#define RT_SQUARE  0x1b5d
#define LT_SQUARE  0x1a5b
#define RT_BRACKET 0x1b7d
#define LT_BRACKET 0x1a7b
#define CNTL_HOME  0x7700
#define CNTL_END   0x7500
#define CNTL_ENTER 0x1c0a
#define CNTL_BS    0x0e7f
```

```

#define HOME 0x4700
#define END 0x4f00
#define s_BS 0x0008
#define BS 0x0e08
#define BACKSPACE 0x0e08
#define s_CR 0x000d
#define CR 0x1c0d
#define ENTER 0x1c0d
#define UP_ARROW 0x4800
#define RIGHT_ARROW 0x4d00
#define LEFT_ARROW 0x4b00
#define DOWN_ARROW 0x5000
#define UP_ARROW_K 0x48e0
#define RIGHT_ARROW_K 0x4de0
#define LEFT_ARROW_K 0x4be0
#define DOWN_ARROW_K 0x50e0
#define F1 0x3b00
#define F2 0x3c00
#define F3 0x3d00
#define F4 0x3e00
#define F5 0x3f00
#define F6 0x4000
#define F7 0x4100
#define F8 0x4200
#define F9 0x4300
#define F10 0x4400
#define SHIFT_TAB 0x0f00
#define SHIFT_HOME 0x4737
#define SHIFT_END 0x4f31
#define SHIFT_INSERT 0x5230
#define SHIFT_DELETE 0x532e
#define SHFT_INSERT 0x5230
#define SHFT_F1 0x5400
#define SHFT_F2 0x5500
#define SHFT_F3 0x5600
#define SHFT_F4 0x5700
#define SHFT_F5 0x5800
#define SHFT_F6 0x5900
#define SHFT_F7 0x5a00
#define SHFT_F8 0x5b00
#define SHFT_F9 0x5c00
#define SHFT_F10 0x5d00
#define SH_R_ARROW 0x4d36
#define SH_L_ARROW 0x4b34
#define SH_U_ARROW 0x4838
#define SH_D_ARROW 0x5032
#define CNTL_F1 0x5e00
#define CNTL_F2 0x5f00
#define CNTL_F3 0x6000
#define CNTL_F4 0x6100
#define CNTL_F5 0x6200
#define CNTL_F6 0x6300
#define CNTL_F7 0x6400
#define CNTL_F8 0x6500
#define CNTL_F9 0x6600
#define CNTL_F10 0x6700
#define CNTL_LEFTA 0x7300
#define CNTL_RIGHTA 0x7400
#define ALT_F1 0x6800
#define ALT_F2 0x6900

```

1-2 Continued.

```
#define ALT_F3      0x6a00
#define ALT_F4      0x6b00
#define ALT_F5      0x6c00
#define ALT_F6      0x6d00
#define ALT_F7      0x6e00
#define ALT_F8      0x6f00
#define ALT_F9      0x7000
#define ALT_F10     0x7100
#define ALT_A       0x1e00
#define ALT_B       0x3000
#define ALT_C       0x2e00
#define ALT_D       0x2000
#define ALT_E       0x1200
#define ALT_F       0x2100
#define ALT_G       0x2200
#define ALT_H       0x2300
#define ALT_I       0x1700
#define ALT_J       0x2400
#define ALT_K       0x2500
#define ALT_L       0x2600
#define ALT_M       0x3200
#define ALT_N       0x3100
#define ALT_O       0x1800
#define ALT_P       0x1900
#define ALT_Q       0x1000
#define ALT_R       0x1300
#define ALT_S       0x1f00
#define ALT_T       0x1400
#define ALT_U       0x1600
#define ALT_V       0x2f00
#define ALT_W       0x1100
#define ALT_X       0x2d00
#define ALT_Y       0x1500
#define ALT_Z       0x2c00
#define CNTL_A      0x1e01
#define CNTL_B      0x3002
#define CNTL_C      0x2e03
#define CNTL_D      0x2004
#define CNTL_E      0x1205
#define CNTL_F      0x2106
#define CNTL_G      0x2207
#define CNTL_H      0x2308
#define CNTL_I      0x1709
#define CNTL_J      0x240a
#define CNTL_K      0x250b
#define CNTL_L      0x260c
#define CNTL_M      0x320d
#define CNTL_N      0x310e
#define CNTL_O      0x180f
#define CNTL_P      0x1910
#define CNTL_Q      0x1011
#define CNTL_R      0x1312
#define CNTL_S      0x1f13
#define CNTL_T      0x1414
#define CNTL_U      0x1615
#define CNTL_V      0x2f16
#define CNTL_W      0x1117
#define CNTL_X      0x2d18
#define CNTL_Y      0x1519
#define CNTL_Z      0x2c1a
```



```

#define K_0      0x0b30
#define K_1      0x0231
#define K_2      0x0332
#define K_3      0x0433
#define K_4      0x0534
#define K_5      0x0635
#define K_6      0x0736
#define K_7      0x0837
#define K_8      0x0938
#define K_9      0x0a39
#define ALT_0    0x8100
#define ALT_1    0x7800
#define ALT_2    0x7900
#define ALT_3    0x7a00
#define ALT_4    0x7b00
#define ALT_5    0x7c00
#define ALT_6    0x7d00
#define ALT_7    0x7e00
#define ALT_8    0x7f00
#define ALT_9    0x8000
#define K_SPACE  0x3920
#define K_EXCLAM 0x0221
#define K_QUOTE  0x2822
#define K_POUND   0x0423
#define K_DOLLAR  0x0524
#define K_PERCENT 0x0625
#define K_AND     0x0826
#define K_APOST   0x2827
#define K_LPAREN  0x0A28
#define K_RPAREN  0x0B29
#define K_STAR    0x092A
#define K_PLUS    0x0D2B
#define K_COMMA   0x332C
#define K_MINUS   0x0C2D
#define K_PERIOD  0x342E
#define K_FSLASH  0x352F
#define K_COLON   0x273A
#define K_SCOLON  0x273B
#define K_LESS    0x333C
#define K_EQUAL   0x0D3D
#define K_GREAT   0x343E
#define K_QUERY   0x353F
#define K_AMPER   0x0340
#define K_A       0x1E61 - 0x20
#define K_B       0x3062 - 0x20
#define K_C       0x2E63 - 0x20
#define K_D       0x2064 - 0x20
#define K_E       0x1265 - 0x20
#define K_F       0x2166 - 0x20
#define K_G       0x2267 - 0x20
#define K_H       0x2368 - 0x20
#define K_I       0x1769 - 0x20
#define K_J       0x246A - 0x20
#define K_K       0x256B - 0x20
#define K_L       0x266C - 0x20
#define K_M       0x326D - 0x20
#define K_N       0x316E - 0x20
#define K_O       0x186F - 0x20
#define K_P       0x1970 - 0x20
#define K_Q       0x1071 - 0x20
#define K_R       0x1372 - 0x20

```

1-2 Continued.

```
#define K_S          0x1F73 - 0x20
#define K_T          0x1474 - 0x20
#define K_U          0x1675 - 0x20
#define K_V          0x2F76 - 0x20
#define K_W          0x1177 - 0x20
#define K_X          0x2D78 - 0x20
#define K_Y          0x1579 - 0x20
#define K_Z          0x2C7A - 0x20
#define K_LBRACK     0x1A5B
#define K_BSLASH     0x2B5C
#define K_RBRACK     0x1B5D
#define K_KARAT      0x075E
#define K_UNDER      0x0C5C
#define K_a          0x1E61
#define K_b          0x3062
#define K_c          0x2E63
#define K_d          0x2064
#define K_e          0x1265
#define K_f          0x2166
#define K_g          0x2267
#define K_h          0x2368
#define K_i          0x1769
#define K_j          0x246A
#define K_k          0x256B
#define K_l          0x266C
#define K_m          0x326D
#define K_n          0x316E
#define K_o          0x186F
#define K_p          0x1970
#define K_q          0x1071
#define K_r          0x1372
#define K_s          0x1F73
#define K_t          0x1474
#define K_u          0x1675
#define K_v          0x2F76
#define K_w          0x1177
#define K_x          0x2D78
#define K_y          0x1579
#define K_z          0x2C7A
```

1-2 Ends.

1-3 The source code listing to ASCII.H.

```
////////////////////////////////////////
//
// ascii.h
//
// ASCII def file
//

#define aNUL          0    //      null \0 delimiter
#define aSOH          1    // ^A - start of heading
#define aSTX          2    // ^B - start of text
#define aETX          3    // ^C - end of text
#define aEOT          4    // ^D - end of transmission
#define aENQ          5    // ^E - inquiry
#define aACK          6    // ^F - affirm acknowledgement
#define aBEL          7    // ^G - audible bell
#define aBS           8    // ^H - backspace
#define aTAB          9    // ^I - horizontal tab
```

```

#define aLF      10  // ^J - line feed
#define aVT      11  // ^K - vertical tab
#define aFF      12  // ^L - form feed
#define aCR      13  // ^M - carriage return
#define aSO      14  // ^N - shift out
#define aSI      15  // ^O - shift in
#define aDCE     16  // ^P - data link escape
#define aDC1     17  // ^Q - device control 1
#define aDC2     18  // ^R - device control 2
#define aDC3     19  // ^S - device control 3
#define aDC4     20  // ^T - device control 4
#define aNAK     21  // ^U - neg acknowledge
#define aSYN     22  // ^V - synchronous idle
#define aETB     23  // ^W - end of transmission
#define aCAN     24  // ^X - cancel
#define aEM      25  // ^Y - end of medium
#define aSUB     26  // ^Z - substitute
#define aESC     27  // escape
#define aFS      28  // file sererator
#define aGS      29  // group seperator
#define aRS      30  // record seperator
#define aUS      31  // unlinked seperator
#define aSPC     32  // space
#define aCODE    94  // ^character indicating printer command follows
#define aHCR     aEOT // Hard carriage return
#define aCENTER  'C'  // code to center line
#define aDOUBLE  'D'  // double strike toggle
#define aEXPAND  'E'  // emphasize toggle
#define aSUPERS  'S'  // superscript toggle
#define aITALIC  'I'  // italics toggle
#define aBOLD    'B'  // bold toggle
#define aTRUE    1    // true
#define aFALSE   0    // false
#define ONE_COL  1    // 1 column format
#define TWO_COL  2    // 2 column format
#define ONE_TOP  3    //
#define TWO_TOP  4    //
#define ONE_BOT  5    //
#define TWO_BOT  6    //
#define TWO_LR   7    //
#define TWO_R    8    //
#define TWO_UR   9    //
#define TWO_TB   10   //
#define VONE_COL 11   // word per chart format
#define XONE_COL 81   // 1 column format
#define XTWO_COL 82   // 2 column format
#define XTHREE_COL 83 // 3 column format
#define XONE_TOP 84   //
#define XTWO_TOP 85   //
#define XTHREE_TOP 86 //
#define XONE_BOT 87   //
#define XTWO_BOT 88   //
#define XTHREE_BOT 89 //
#define XTHREE_LR 90  //
#define XTHREE_R  91  //
#define XTHREE_P1 92  //
#define XTHREE_P2 93  //
#define XTHREE_TB 94  //
#define XTHREE_UR 95  //
#define XTHREE_2T 96  //
#define XTHREE_2B 97  //

```

1-3 Ends.

1-4 The source code listing to TSTRUCT.H.

```
////////////////////////////////////
//
// tstruct.h
//

#ifndef IMAGE
#define IMAGE unsigned int
#endif

#ifndef ULONG
#define ULONG unsigned long
#endif

#ifndef USHORT
#define USHORT unsigned short
#endif

#ifndef UINT
#define UINT unsigned int
#endif

#ifndef UINTP
#define UINTP UINT *
#endif

#ifndef UCHAR
#define UCHAR unsigned char
#endif

#ifndef UCHARP
#define UCHARP UCHAR *
#endif

/*
 * structures
 */

////////////////////////////////////
//
// Interface Structure List
//

////////////////////////////////////

#define LOTUS_ITEM_MAX 20
#define MENUBAR_ITEM_MAX 10

////////////////////////////////////
//
// Structure for Lotus Style Window Interface
//

typedef struct {
    int number;           // number of LOTUS objects
    char *name[LOTUS_ITEM_MAX]; // pointer to item name
    char *explain[LOTUS_ITEM_MAX]; // pointer to item explanation
    int lot_map[LOTUS_ITEM_MAX][2]; // map for lotus item highlights
}
```

```

    int lotus_item;           // highlight and
    int old_lotus;           // item selection data
    int lotus_open;          // status of lotus window
    unsigned int imgbuf[160]; // top two rows screen image
} LOTUS_CLASS;

////////////////////////////////////
//
// Structure for Lotus Style Window Interface
//

typedef struct {
    int number;           // number of MENUBAR objects
    char *name[MENUBAR_ITEM_MAX]; // pointer to item name
    int mb_map[MENUBAR_ITEM_MAX][2]; // map for menubar item highlights
    int key_list[MENUBAR_ITEM_MAX];
    int menubar_item; // highlight and
    int old_menubar; // item selection data
    int menubar_open; // status of lotus window
    int si_attr;      // item attribute
    int sinv_attr;    // inverse attribute
    int sk_attr;      // highlight key attribute
    int first_time;   // first time
    unsigned int imgbuf[160]; // top two rows screen image
} MENUBAR_CLASS;

typedef struct {
    int ul_row;           /* upper left row */
    int ul_col;           /* upper left column */
    int lr_row;           /* lower right row */
    int lr_col;           /* lower right column */
    unsigned int img_size; /* window img size */
    unsigned int *img_ptr; /* pointer scrn image */
    unsigned int *wind_ptr; /* pointer scrn image */
    int box_type;         /* border selection */
    int attr;             /* window attribute */
    int visible;          /* window on */
    int top_offset;       /* col offset title */
    int top_length;       /* length title str */
    int show_top;         /* display title */
    int bot_offset;       /* col offset title */
    int bot_length;       /* length title str */
    int show_bot;         /* display title */
    char *t_title;        /* ptr to t title str */
    char *b_title;        /* ptr to b title str */
} WIND;

typedef struct {
    int ul_row;           /* upper left row */
    int ul_col;           /* upper left column */
    int lr_row;           /* lower right row */
    int lr_col;           /* lower right column */
    unsigned int *img_ptr; /* pointer scrn image */
    int box_type;         /* border selection */
    int attr;             /* window attribute */
    int visible;          /* window on */
    int show_top;         /* display title */
    char *t_title;        /* ptr to t title str */
} TWIND;

```

1-4 Continued.

```
typedef struct {
    unsigned char media_descr;    /* media descriptor byte */
    unsigned int clust_avail;     /* # of free clusters on disk */
    unsigned int clust_total;     /* total # of clusters on disk */
    unsigned int sec_p_clust;     /* # of sectors per cluster */
    unsigned int bytes_p_sec;     /* # of bytes per sector */
} DSKINFO;

typedef struct {
    int mode;                    /* video mode */
    int row_width;               /* columns per row */
    int page;                     /* video page */
    unsigned int *scrn;          /* pointer to video RAM */
} VIDEO;

typedef struct {
    int ul_row;                  /* upper left row */
    int ul_col;                  /* upper left column */
    int lr_row;                  /* lower right row */
    int lr_col;                  /* lower right column */
    unsigned int *image;         /* pointer to scrn image */
} RECT;

typedef struct {
    int row;                     /* cursor row */
    int column;                  /* cursor column */
} CUR_LOCATION;

typedef struct {
    int status;                  /* pen down or up */
    int pix_col;                 /* pixel column */
    int pix_row1;                /* pixel row */
    int pix_row2;                /* pixel row */
    int ch_row;                  /* character row */
    int ch_col;                  /* character column */
} LIGHT_PEN;

/*
 * defines for wrbox
 */

#define S_S_S_S 0
#define S_S_D_D 1
#define D_D_S_S 2
#define D_D_D_D 3

/*
 * defines for mkAttr
 */

#define BLACK 0
#define BLUE 1
#define GREEN 2
#define CYAN 3
#define RED 4
#define MAGENTA 5
#define BROWN 6
#define WHITE 7
```

```

#define NORMAL      7
#define REVERSE 112

#define ON_INTENSITY    8
#define OFF_INTENSITY 0
#define ON_BLINK        128
#define OFF_BLINK       0

/*
 * defines for scroll routines
 */

#define UP_SCROLL 6
#define DOWN_SCROLL 7

//
// defines for printer routines
//

#define PRINT_TIME_OUT 1
#define IO_ERROR      4
#define PRINT_SELECTED 8
#define OUT_OF_PAPER   16
#define ACKNOWLEDGE    32
#define PRINT_NOT_BUSY 64

//
// defines for flush kb buffer and get char
//

#define ON_ECHO_CTRL      1 /* on char echo and control-c enabled */
#define OFF_ECHO_CTRL_C 7 /* off echo and control-c disabled */
#define OFF_ECHO          8 /* off echo and control-c enabled */

//
// defines for kb shift status
//

#ifdef DOS_PROG

#define RIGHT_SHIFT 1
#define LEFT_SHIFT  2
#define CTRL_PRESS  4
#define ALT_PRESS    8
#define SCROLL_LOCK 16
#define NUM_LOCK     32
#define CAPS_LOCK    64
#define INSERT_ON    128

#endif

//
// defines for MENU routines
//

#define CENTER      0xff
#define NUMBERED    1
#define RESETROW    2

```

1-4 Continued.

```
//
// defines for kb_edit
//

#define UPPER_LOWER 0
#define UPPER      1
#define LOWER      2
#define NAME       3


//
// defines for mouse routines
//

#define LEFTBUTTON 1
#define RIGHTBUTTON 2
#define CNTRBUTTON 4


//
// io defines
//

#ifndef BCOS2_COMP
#ifndef DOS_PROG
#define S_IFMT 0xF000 // file type mask
#define S_IFDIR 0x4000 // directory
#define S_IFIFO 0x1000 // FIFO special
#define S_IFCHR 0x2000 // character special
#define S_IFBLK 0x3000 // block special
#define S_IFREG 0x8000 // or just 0x0000, regular
#define S_IREAD 0x0100 // owner may read
#define S_IWRITE 0x0080 // owner may write
#define S_IXEC 0x0040 // owner may execute <directory search>
#define O_RDONLY 1
#define O_WRONLY 2
#define O_RDWR 4
// Flag values for open only
#define O_CREAT 0x0100 // create and open file
#define O_TRUNC 0x0200 // open with truncation
#define O_EXCL 0x0400 // exclusive open

#endif
#endif
```

1-4 Ends.

Building your libraries and demonstration programs

I used a command file (BUILDLIB.CMD, FIG. 1-5) to build the OS/2 Full Screen library and compile and link the demonstration programs. This is the command file for the IBM C compiler. Figure 1-5 presents the source code listing to BUILDLIB.CMD.

1-5 The source code listing to BUILDLIB.CMD.

```
del *.lib
del *.bak
```



```

del *.obj
del *.exe

icc /DOS2_PROG /C /Ss+ /Sp keyboard.c
lib os2text +keyboard;

icc /DOS2_PROG /C /Ss+ /Sp make.c
lib os2text +make;

icc /DOS2_PROG /C /Ss+ /Sp screen.c
lib os2text +screen;

icc /DOS2_PROG /C /Ss+ /Sp cursor.c
lib os2text +cursor;

icc /DOS2_PROG /C /Ss+ /Sp rect.c
lib os2text +rect;

icc /DOS2_PROG /C /Ss+ /Sp mouse.c
lib os2text +mouse;

icc /DOS2_PROG /C /Ss+ /Sp window.c
lib os2text +window;

icc /DOS2_PROG /C /Ss+ /Sp printer.c
lib os2text +printer;

icc /DOS2_PROG /Ss+ /Sp prog2-1.c os2text.lib
icc /DOS2_PROG /Ss+ /Sp prog2-2.c os2text.lib
icc /DOS2_PROG /Ss+ /Sp prog2-3.c os2text.lib
icc /DOS2_PROG /Ss+ /Sp prog2-4.c os2text.lib
icc /DOS2_PROG /Ss+ /Sp prog2-5.c os2text.lib
icc /DOS2_PROG /Ss+ /Sp prog3-1.c os2text.lib
icc /DOS2_PROG /Ss+ /Sp prog4-1.c os2text.lib
icc /DOS2_PROG /Ss+ /Sp prog4-2.c os2text.lib
icc /DOS2_PROG /Ss+ /Sp prog5-1.c os2text.lib
icc /DOS2_PROG /Ss+ /Sp prog5-2.c os2text.lib
icc /DOS2_PROG /Ss+ /Sp prog5-3.c os2text.lib
icc /DOS2_PROG /Ss+ /Sp prog6-1.c os2text.lib
icc /DOS2_PROG /Ss+ /Sp prog6-2.c os2text.lib
icc /DOS2_PROG /Ss+ /Sp prog6-3.c os2text.lib
icc /DOS2_PROG /Ss+ /Sp prog6-4.c os2text.lib
icc /DOS2_PROG /Ss+ /Sp prog6-6.c os2text.lib
icc /DOS2_PROG /Ss+ /Sp prog6-7.c os2text.lib
icc /DOS2_PROG /Ss+ /Sp prog6-8.c dfinst1.c os2text.lib
icc /DOS2_PROG /Ss+ /Sp prog7-1.c os2text.lib
icc /DOS2_PROG /Ss+ /Sp prog7-2.c os2text.lib

del *.bak
del *.obj

```

1-5 Ends.

I used a batch file (BUILDLIB.BAT, FIG. 1-6) to build the DOS library and compile and link the DOS demonstration programs. Figure 1-6 presents the source code listing to BUILDLIB.BAT.

1-6 The source code listing to BUILDLIB.BAT.

```

del *.lib
bcc -c -ml -DDOS_PROG screen.c

```

1-6 Continued.

```
bcc -c -ml -DDOS_PROG make.c
bcc -c -ml -DDOS_PROG window.c
bcc -c -ml -DDOS_PROG keyboard.c
bcc -c -ml -DDOS_PROG cursor.c
bcc -c -ml -DDOS_PROG rect.c
bcc -c -ml -DDOS_PROG mouse.c
bcc -c -ml -DDOS_PROG printer.c
tasm /mx /Dmdl=3 kb_stat.asm

tlib dostext +screen.obj
tlib dostext +make.obj
tlib dostext +window.obj
tlib dostext +keyboard.obj
tlib dostext +cursor.obj
tlib dostext +rect.obj
tlib dostext +mouse.obj
tlib dostext +printer.obj
tlib dostext +kb_stat.obj

bcc -ml -DDOS_PROG prog2-1.c dostext.lib
bcc -ml -DDOS_PROG prog2-2.c dostext.lib
bcc -ml -DDOS_PROG prog2-3.c dostext.lib
bcc -ml -DDOS_PROG prog2-4.c dostext.lib
bcc -ml -DDOS_PROG prog2-5.c dostext.lib
bcc -ml -DDOS_PROG prog3-1.c dostext.lib
bcc -ml -DDOS_PROG prog4-1.c dostext.lib
bcc -ml -DDOS_PROG prog4-2.c dostext.lib
bcc -ml -DDOS_PROG prog5-1.c dostext.lib
bcc -ml -DDOS_PROG prog5-2.c dostext.lib
bcc -ml -DDOS_PROG prog5-3.c dostext.lib
bcc -ml -DDOS_PROG prog6-1.c dostext.lib
bcc -ml -DDOS_PROG prog6-2.c dostext.lib
bcc -ml -DDOS_PROG prog6-3.c dostext.lib
bcc -ml -DDOS_PROG prog6-4.c dostext.lib
bcc -ml -DDOS_PROG prog6-5.c dostext.lib
bcc -ml -DDOS_PROG prog6-6.c dostext.lib
bcc -ml -DDOS_PROG prog6-7.c dostext.lib
bcc -ml -DDOS_PROG prog6-8.c dostext.lib
bcc -ml -DDOS_PROG prog7-1.c dostext.lib
bcc -ml -DDOS_PROG prog7-2.c dostext.lib

del *.obj
del *.bak
del *.lst
```

1-6 Ends.

I used a command file (BUILDLIB.CMD, FIG. 1-7) to build the OS/2 Full Screen library and compile and link the demonstration programs. This is the command file for the Borland C++ for OS/2 compiler. Figure 1-7 presents the source code listing to BUILDLIB.CMD.

1-7 The source code listing to BUILDLIB.CMD.

```
del *.obj
del *.lib
del *.bak

bcc -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP -c keyboard.c
bcc -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP -c make.c
```

```

bcc -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP -c screen.c
bcc -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP -c cursor.c
bcc -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP -c rect.c
bcc -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP -c mouse.c
bcc -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP -c window.c
bcc -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP -c printer.c

tlib os2text +screen
tlib os2text +make
tlib os2text +keyboard
tlib os2text +cursor
tlib os2text +rect
tlib os2text +mouse
tlib os2text +window
tlib os2text +printer

bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog2-1.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog2-2.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog2-3.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog2-4.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog2-5.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog3-1.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog4-1.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog4-2.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog5-1.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog5-2.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog5-3.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog6-1.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog6-2.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog6-3.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog6-4.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog6-5.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog6-6.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog6-7.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog6-8.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog7-1.c
os2text.lib
bcc -Ld:\bcos2\lib -Id:\bcos2\include -DOS2_PROG -DBCOS2_COMP prog7-2.c
os2text.lib

del *.obj
del *.bak

```

1-7 Ends.

Summary

This chapter presented the foundation function prototype (TPROTO.H, FIG. 1-1) file. This file summarizes the uniform OS/2 and Full Screen and DOS API presented in this book. This uniform API facilitates the development of OS/2 Full Screen and DOS applications.

Include files for structures which are used in the API and key defines are also presented (KEYBOARD.H, FIG. 1-2, ASCII.H, FIG. 1-3 and TSTRUCT.H, FIG. 1-4).

Finally, a command file (BUILDLIB.CMD, FIG. 6-5) demonstrates one way to build the OS/2 Full Screen version of the book's API using the IBM C Compiler. The batch file (BUILDLIB.BAT, FIG. 6-6) presents one way to build the DOS API. The command file for building the OS/2 library with Borland's C++ for OS/2 compiler is presented in FIG. 6-7.

Chapter 2 introduces the keyboard management demonstration function prototypes and demonstration programs.

2

Keyboard management demonstration programs

This chapter presents five programs that demonstrate the use of five keyboard management functions. These functions form a solid foundation for all of a program's keyboard input needs.

Table 2-1 presents the function prototypes for the keyboard related functions.

Table 2-1 Keyboard function prototypes.

int	kb_edit(char	*response,
	int	row,
	int	column,
	int	dlen,
	int	opt,
		UCHARattr);
int	kb_read(void);	
int	kb_status(void);	
char	kb_char(void);	
UCHAR	kb_scan(void);	

Keyboard function descriptions

Function `kb_read()`

Usage `key= kb_read();`
 where
 key is a 16-bit int

Remarks Function `kb_read()` stops program execution and waits for a key press. When a key is pressed program execution continues. The 16-bit key code is returned to key. The 16-bit defines for the returned key are in `KEYBOARD.H` (FIG. 1-2).

Function `kb_scan()`

Usage `scan= kb_scan();`
 where
 scan is an 8-bit unsigned char

Remarks Function `kb_scan()` stops program execution and waits for a key press. When a key is pressed program execution continues. The 8-bit scan code is returned to scan.

Function `kb_char()`

Usage `ch= kb_char();`
 where
 ch is an 8-bit char

Remarks Function `kb_char()` stops program execution and waits for a key press. When a key is pressed program execution continues. The 8-bit char code is returned to ch.

Function `kb_status()`

Usage `key= kb_status();`
 where
 key is a 16-bit int

Remarks Function `kb_status()` does not stop program execution and reports when a key is pressed. If no key is pressed a value of 0 is returned. If a key is pressed, the 16-bit key code is returned to key. The 16-bit defines for the returned key are presented in `KEYBOARD.H` (FIG. 1-2).

Function kb__edit(...)

Usage key= kb__edit(*response, row, column, dlen, opt, attr);

where

key is a 16-bit int

response is a pointer to a char buffer

row is a 16-bit int

column is a 16-bit int

dlen is a 16-bit int

opt is a 16-bit int

attr is an unsigned char

Remarks Function kb__edit(...) allows you to enter a string of text from the keyboard. The function terminates using a Tab, Enter, or Esc character. The terminating key press is returned in the 16-bit int, key. If you press the insert key, kb__edit goes into the insert mode, otherwise the default mode is overlay. The text is returned to the buffer pointed to by response. You can set the row and column where the input will begin. The length of the buffer is also set. The attribute of the entry text is also controlled. The dopt variable controls the case of the text. UPPER_CASE means that all text will appear as uppercase. LOWER_CASE means that all text will appear as lowercase. UPPER_LOWER means that the person entering data can select the case.

Stopping program execution & reading a 16-bit key code

Figure 2-1 presents the source code listing for PROG2-1.C. This program demonstrates the use of function kb__read().

2-1 The source code listing to PROG2-1.C.

```
////////////////////////////////////
//
// prog2-1.c
//
// Demonstrates:
//
//     kb__read()
//     -----
//     Stops program execution and
//     reads 16-bit key code
//
//
////////////////////////////////////

////////////////////////////////////
//
// Compiler includes here
```

2-1 Continued.

```
//
#include <stdio.h>

////////////////////////////////////
//
// Library includes here
//

#include "tproto.h"

////////////////////////////////////
//
// main function
//

void main()
{
// 16-bit scan and char code

int code;

//
// print program title
//

printf("\n\nDemonstration of function kb_read()\n");
printf("Press F10 to exit\n\n");

//
// print 16-bit key codes until
// the F10 key is pressed
//

do {
// stop program and wait for key press
// 16-bit key code => code

code= kb_read();

// print 16-bit key code to the screen

printf("Key Code= 0x%04X\n", code);

// repeat until F10 is pressed

} while(code != F10);

}

//
////////////////////////////////////
```

2-1 Ends.

Stopping program execution & reading an 8-bit character code

Figure 2-2 presents the source code listing for PROG2-2.C. This program demonstrates the use of function `kb__char()`.

2-2 The source code listing to PROG2-2.C.

```
////////////////////////////////////////
//
// prog2-2.c
//
// Demonstrates:
//
//     kb_char()
//     -----
//     Stops program execution and
//     reads 8-bit character code
//
////////////////////////////////////////

////////////////////////////////////////
//
// Compiler includes here
//

#include <stdio.h>

////////////////////////////////////////
//
// Library includes here
//

#include "tproto.h"

////////////////////////////////////////
//
// main function
//

void main()
{
// 8-bit char code
UCHAR code;

//
// print program title
//

printf("\n\nDemonstration of function kb_char()\n");
printf("Press X or x to exit\n\n");

//
// print 8-bit char codes and char until
// the x or X key is pressed
//

do {
// stop program and wait for key press
// 8-bit char code => code

code= kb_char();

// print 8-bit char code to the screen

printf("Char Code= 0x%02X   %c\n", code, code);
```

2-2 Continued.

```
        // repeat until x or X is pressed

        } while((code != 'x') && (code != 'X'));
    }

    //
    //////////////////////////////////////
```

2-2 Ends.

Stopping program execution & reading an 8-bit scan code

Figure 2-3 presents the source code listing for PROG2-3.C. This program demonstrates the use of function `kb_scan()`.

2-3 The source code listing to PROG2-3.C.

```
////////////////////////////////////
//
// prog2-3.c
//
// Demonstrates:
//
//     kb_scan()
//     -----
//     Stops program execution and
//     reads 8-bit scan code
//
////////////////////////////////////
////////////////////////////////////
//
// Compiler includes here
//

#include <stdio.h>

////////////////////////////////////
//
// program defines
//

#define ALT_X_SCAN_CODE 0x2d

////////////////////////////////////
//
// Library includes here
//

#include "tproto.h"

////////////////////////////////////
//
// main function
//

void main()
{
```

```

// 8-bit char code
UCHAR code;

//
// print program title
//

printf("\n\nDemonstration of function kb_scan()\n");
printf("Press ALT_X to exit\n\n");

//
// print 8-bit scan codes until
// the Alt-X key combination is pressed
//

do {
    // stop program and wait for key press
    // 8-bit scan code => code

    code= kb_scan();

    // print 8-bit scan code to the screen

    printf("Scan Code= 0x%02X\n", code);

    // repeat until x or X is pressed

    } while(code != ALT_X_SCAN_CODE);
}

//
////////////////////////

```

2-3 Ends.

Not stopping program & retrieving a 16-bit key code

Figure 2-4 presents the source code listing for PROG2-4.C. This program demonstrates the use of function `kb__status()`.

2-4 The source code listing to PROG2-4.C.

```

////////////////////////////////////
//
// prog2-4.c
//
// Demonstrates:
//
//     kb__status()
//     -----
//     Does not stops program execution
//     reports 16-bit key code after a
//     key press
//
////////////////////////////////////

////////////////////////////////////
//
// Compiler includes here
//

```

2-4 Continued.

```
#include <stdio.h>

////////////////////////////////////
//
// Library includes here
//

#include "tproto.h"

////////////////////////////////////
//
// main function
//

void main()
{
    // 16-bit scan and char code

    int code;

    //
    // print program title
    //

    printf("\n\nDemonstration of function kb_status()\n");
    printf("Press any key to start program\n");
    printf("Press F10 to stop program execution\n\n");

    //
    // wait for key press
    //

    kb_read();

    //
    // print 16-bit key codes until
    // the F10 key is pressed
    //

    do {
        // do not stop program and check for key press
        // 16-bit key code => code

        code= kb_status();

        // if a key has been pressed then print it
        // to the screen

        if(code != 0) {

            // print 16-bit key code to the screen

            printf("\n\nKey Code= 0x%04X\n", code);

            // and wait for another key press to continue

            printf("Press any key to restart program\n");

            kb_read();
        }
    }
}
```

```

    }

    // else print message to screen

    else {
        printf("No key press...");
    }

    // repeat until F10 is pressed

    } while(code != F10);

}

//
//

```

2-4 Ends.

Retrieving a character string from the keyboard

Figure 2-5 presents the source code listing for PROG2-5.C. This program demonstrates the use of function `kb__edit(...)`.

2-5 The source code listing to PROG2-5.C.

```

//
//
// prog2-5.c
//
// Demonstrates
//
// kb_edit()
// -----
// Stops program execution and
// and reads a character string
// from the keyboard at a specified
// row and column location.
//
//
//
//
//
// Compiler includes here
//
#include <stdio.h>
#include <memory.h>

//
// Library includes here
//

#include "tproto.h"

//
//
// main function
//

```

2-5 Continued.

```
void main()
{
char buffer[60];          // 60 byte string buffer
int  code;                // 16-bit return code

    //
    // initialize screen
    //
    // Required by DOS programs using the library
    // Not required by OS/2 programs
    //
    // An empty function has been provided to
    // reduce the number of conditional compilation
    // considerations
    //
    // Use conditional compilation if you wish
    //

#ifdef DOS_PROG

    scrn_init();

#endif

    //
    // clear the screen
    //

    scrn_clear();

    //
    // initialize buffer
    //

    memset(buffer, 0, 60);

    //
    // print program title
    //

    scrn_write(0, 0, 0, "kb_edit(...) demonstration", 7);

    //
    // data entry field name
    //

    scrn_write(2, 0, 0, "Enter Full Name: ", 7);

    //
    // read keyboard string
    //

    code= kb_edit(buffer,          // buffer receives data
            2,                    // row start
            16,                   // column start
            60,                   // length of entry field
            UPPER_LOWER,         // allow upper and lower case
            7);                  // attribute
}
```

```

// process return code
//
// if the return code is Escape
if(code == ESC) {
    scrn_write(4, 0, 0, "Escape Key... No action", 7);
}

// otherwise Enter key press returned
else {
    scrn_write(4, 0, 0, "Enter Key... action", 7);
    scrn_write(5, 0, 0, "Data entered: ", 7);
    scrn_write(5, 14, 0, buffer, 7);
}

//
// wait for key press
//

kb_read();

//
// clear the screen
//

scrn_clear();

}

//
////////////////////

```

2-5 Ends.

Summary

This chapter presented programs which demonstrate the use of foundation keyboard handling routines. You learned how to

- Stop program execution and retrieve a 16-bit key press.
- Not stop program execution and see if a key had been pressed.
- Stop program execution and retrieve the scan code of the key pressed.
- Stop program execution and retrieve the char code of the key pressed.
- Enter a string of characters from a specified row and column screen location.

Chapter 3 introduces management function for controlling the text cursor.

3

Text cursor management demonstration programs

This chapter presents one demonstration program that illustrates the use of most of the cursor management functions presented in this chapter.

Table 3-1 presents the function prototypes for cursor related functions.

**Table 3-1 Cursor
function prototypes.**

void	cu_get_loc(int *, int *);
int	cu_get_shape(void);
void	cu_move(int, int);
void	cu_relative_move(int, int);
void	cu_remove(void);
void	cu_display(void);
void	cu_save_size(void);
void	cu_rest_size(void);
void	cu_set_size(int, int);
void	cu_save_loc(void);
void	cu_set_shape(int);
void	cu_rest_loc(void);

Cursor function descriptions

Function `cu__get__loc(...)`

Usage `cu__get__loc(*row, *col);`
 where
 `row` is a pointer to an integer
 `col` is a pointer to an integer

Remarks Function `cu__get__loc(...)` returns the current row and column cursor locations via the integer pointers present in the parameter list.

Function `cu__get__shape()`

Usage `shape= cu__get__shape();`
 where
 `shape` is an integer that holds the cursor begin and end data

Remarks Function `cu__get__shape()` is a convenient way to save the cursor's shape.

Function `cu__set__shape(...)`

Usage `cu__set__shape(shape);`
 where
 `shape` is an int

Remarks Function `cu__set__shape(...)` should be invoked after the shape has been saved in `shape` using function `cu__get__shape(...)`.

Function `cu__move(...)`

Usage `cu__move(row, col);`
 where
 `row` is an int
 `col` is an int

Remarks Function `cu__move` moves the text cursor to a specified row and column location.

Function `cu__relative__move(...)`

Usage `cu__relative__move(row__offset, col__offset);`
 where
 `row__offset` is an integer

`col__offset` is an integer

Remarks Function `cu__relative__move(...)` is a convenient way to move the cursor a specified number of rows and columns relative to the cursor's current position.

Function: `cu__remove()`

Usage `cu__remove();`

Remarks Function `cu__remove()` removes the blinking text cursor from the screen display. This function will prove invaluable in creating classy looking interface screens and certain dialog boxes.

Function: `cu__display()`

Usage `cu__display();`

Remarks Function `cu__display()` displays the blinking text cursor after it has been removed.

Function `cu__save__size()`

Usage `cu__save__size();`

Remarks Function `cu__save__size()` is a convenient way to save the cursor's shape. It differs from function `cu__get__shape()` in that it saves the cursor's size (or shape) to a static variable.

Function `cu__rest__size()`

Usage `cu__rest__size();`

Remarks Function `cu__rest__size()` is a convenient way to save the cursor's shape. It should be invoked after the cursor shape has been saved using `cu__save__size()`.

Function `cu__set__size(...)`

Usage `cu__set__size(start, stop);`

where

`start` is an int

`stop` is an int

Remarks This function is useful for changing the cursor's size.

Function `cu__save__loc()`

Usage `cu__save__loc();`

Remarks This function is a very convenient way to save the cursor's current location to a static variable.

Function `cu__rest__loc()`

Remarks This function is a very convenient way to restore the cursor's previously saved location using the `cu__save__loc()` function.

Manipulating the cursor location, size, and visibility

Figure 3-1 presents the source code listing to PROG3-1.C. This program demonstrates the use of many of the cursor management functions previously presented in the chapter.

3-1 The source code listing to PROG3-1.C.

```
////////////////////////////////////////
//
// prog3-1.c
//
// Demonstrates:
//
//   cu_save_loc()
//   -----
//   Save the cursor's current row
//   and column location
//
//   cu_rest_loc()
//   -----
//   Restores the cursor to the
//   location previously saved by
//   invoking cu_save_loc()
//
//   cu_save_size()
//   -----
//   Save the cursor's current size
//
//   cu_rest_size()
//   -----
//   restores the cursor's current
//   size
//
//   cu_remove()
//   -----
//   Removes the cursor from the
//   screen
//
//   cu_display()
//   -----
//   Displays the cursor on the
//   screen
//
//   cu_move()
//   -----
//   Moves the cursor to a specified
//   row and column location.
```

```

//
////////////////////////////////////
////////////////////////////////////
//
// Compiler includes here
//

#include <stdio.h>

////////////////////////////////////
//
// Library includes here
//

#include "tproto.h"

////////////////////////////////////
//
// main function
//

void main()
{
    //
    // initialize the screen
    //
    scrn_init();

    //
    // save the current cursor location
    //

    cu_save_loc();

    //
    // save the current cursor size
    //

    cu_save_size();

    //
    // save the screen
    //

    scrn_save();

    //
    // clear the screen
    //

    scrn_clear();

    //
    // move the cursor to row 0
    //                      column 0
    //

    cu_move(0, 0);

    //

```

3-1 Continued.

```
// print program title
//

printf("Demonstration of Cursor management functions.\n");
printf("Press any key to remove the cursor.\n");

//
// stop program and wait for key press
//

kb_read();

//
// remove the cursor from the screen
//

cu_remove();

//
// print message
//

printf("Press any key to display the cursor.\n");

//
// stop program and wait for key press
//

kb_read();

//
// display the cursor
//

cu_display();

//
// print message
//

printf("Press any key to change the cursor shape.\n");

//
// stop program and wait for key press
//

kb_read();

cu_set_size(0,14);

//
// print message
//

printf("Press any key to restore calling OS program\n");
printf("while restoring original cursor location and size.\n");

//
// stop program and wait for key press
```

```

//
kb_read();

//
// restore the screen
//

scrn_restore();

//
// restore cursor location
//

cu_rest_loc();

//
// restore prevuisly saved cursor size
//

cu_rest_size();

}

//
////////////////////////////////////

```

3-1 Ends.

Summary

This chapter presents a program that demonstrates the use of foundation cursor management functions. You learned how to:

- Move the text cursor.
- Move the cursor relevant to its current location.
- Remove the text cursor from the display.
- Display a previously removed text cursor.
- Save the cursor position.
- Restore the cursor position.
- Get the cursor location.
- Get the cursor shape.
- Set the cursor size.

Chapter 4 introduces foundation screen management functions.

4

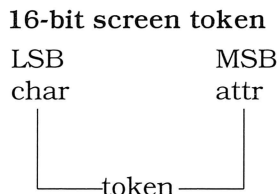
Screen management demonstration programs

This chapter contains two demonstration programs that illustrate how most of the screen management functions are utilized.

For purposes of both the OS/2 Full Screen Library and the DOS library, an 8-bit unsigned char screen attribute is composed of four qualities. These qualities are:

- Foreground color
- Background color
- Foreground intensity
- Foreground blink

The screen token is a 16-bit integer where the LSB is an 8-bit character and the MSB is an 8-bit unsigned char attribute.



Note that function `scrn__init()` is only an empty shell function in the OS/2 Full Screen library. This has been done to minimize the differences between the OS/2 Full Screen library and the DOS library. Note that the `make (mk__)` function descriptions follow the screen function descriptions.

Table 4-1 presents the function prototypes for the screen and make related functions, while Table 4-2 presents make function prototypes.

Table 4-1 Screen function prototypes.

```

void  scrn_write(int      row,
                  int      col,
                  int      len,
                  char     *str,
                  UCHAR   attr);

void  scrn_init(void);
int   scrn_read_char(unsigned short, unsigned short);
void  scrn_save(void);
void  scrn_restore(void);
void  scrn_clear(void);
void  scrn_attr(int, int, int, unsigned char);
void  scrn_change_attr(unsigned char);
void  scrn_char(int row, int col, char ch, UCHAR attr);
void  scrn_repeat_char(int      row,
                       int      col,
                       int      len,
                       char     ch,
                       UCHAR   attr);

void  scrn_chr(int, int, int);

```

Table 4-2 Make function prototypes.

```

UCHAR mk_attr(UCHAR fore,
               UCHAR back,
               UCHAR intensity,
               UCHAR blink);

UCHAR mk_attr_intense(UCHAR attr);
UCHAR mk_attr_intense_blink(UCHAR attr);
UCHAR mk_attr_inverse(UCHAR attr);
UCHAR mk_attr_blink(UCHAR attr);
int   mk_token(UCHAR lsb, UCHAR msb);

```

Screen function descriptions

Function `scrn__init()`

Usage `scrn__init();`

Remarks This function is used to initialize the pointer to direct screen memory in the DOS library. As the OS/2 library does not write

directly to the screen, function `scrn__init()` is an empty shell in the OS/2 library.

Function `scrn__save()`

Usage `scrn__save();`

Remarks This function saves the screen to a static buffer. It provides a quick and easy way to save the text screen image.

Function `scrn__restore()`

Usage `scrn__restore();`

Remarks This function restores a screen image that has been previously saved using the `scrn__save()` function.

Function `scrn__clear()`

Usage `scrn__clear();`

Remarks This function provides a quick and easy way to clear the text screen. It clears the screen using the NORMAL (7) screen attribute. The NORMAL attribute designates WHITE for the foreground color, BLACK for the background color, sets foreground INTENSITY off and foreground BLINK off.

Function `scrn__write(...)`

Usage `scrn__write(row, col, len, *str, attr);`

where

`row` is an int

`col` is an int

`len` is an int

`str` is a pointer to a character buffer

`attr` is an unsigned char

Remarks This function writes a string of characters to the screen at a designated row and column screen location using a specified screen attribute. Note that if the `len` variable is 0 the entire string (NULL terminated) will be printed. If `len` is greater than 0 then only `len` number of bytes will be printed.

Function `scrn__char(...)`

Usage `scrn__char(row, col, ch, attr);`

where

`row` is an int

col is an int
ch is a character
attr is an unsigned character

Remarks This function writes a character to the screen at a specified row and column location using a designated screen attribute.

Function `scrn_chr(...)`

Usage `scrn_chr(row, col, token);`
where
row is an int
col is an int
token is an int

Remarks This function writes a 16-bit screen token to the screen at a specified row and column screen location.

Function `scrn_repeat_char(...)`

Usage `scrn_repeat_char(row, col, length, ch, attr);`
where
row is an int
col is an int
length is an int
ch is a character
attr is an unsigned character

Remarks This function repeats a singular character length number of times using a designated attribute starting at a specified row and column screen location.

Function `scrn_attr(...)`

Usage `scrn_attr(row, col, length, attribute);`
where
row is an int
col is an int
length is an int
attribute is an unsigned char

Remarks This function alters length number of screen attributes starting at a designated row and column screen location without altering screen characters. Function `scrn_attr(...)` proves very useful for creating highlight bars in menus.

Function `scrn__change__attr(...)`

Usage `scrn__change__attr(attr);`
 where
 `attr` is an unsigned char

Remarks This function changes all the attributes of a screen to a specified value without altering the character values appearing on the screen.

Function `scrn__read__char(...)`

Usage `token= scrn__read__char(row, col);`
 where
 `token` is an int
 `row` is an unsigned short
 `col` is an unsigned short

Remarks This function returns a 16-bit screen token from a specified row and column screen location. The screen character value is placed in the LSB of the token and the screen character attribute is placed in the token's MSB.

Make functions description

Function `mk__attr(...)`

Usage `attr= mk__attr(RED,`
 `WHITE,`
 `OFF__INTENSITY,`
 `OFF__BLINK);`
 where
 `attr` is an unsigned char
 `RED` is an unsigned char
 `WHITE` is an unsigned char
 `OFF__INTENSITY` is an unsigned char
 `OFF__BLINK` is an unsigned char

Remarks This function returns an 8-bit screen attribute. Note that the four parameters used in the example are defined in `TSTRUCT.H` (FIG. 1-4).

Function `mk__attr__intense(...)`

Usage `intense= mk__attr__intense(attr);`
 where

intense is an unsigned char
attr is an unsigned char

Remarks This function receives a screen attribute previously created using the `mk_attr(...)` function and turns foreground intensity on.

Function `mk_attr_inverse(...)`

Usage `inverse= mk_attr_inverse(attr);`
where
inverse is an unsigned char
attr is an unsigned char

Remarks This function receives a screen attribute previously created using the `mk_attr(...)` function and reverses the foreground and background colors.

Function `mk_attr_blink(...)`

Usage `blink= mk_attr_blink(attr);`
where
blink is an unsigned char
attr is an unsigned char

Remarks This function receives a screen attribute previously created using the `mk_attr(...)` function and turns the foreground blink on.

Function `mk_attr_intense_blink(...)`

Usage `intense_blink= mk_attr_intense_blink(attr);`
where
intense_blink is an unsigned char
attr is an unsigned char

Remarks This function receives a screen attribute previously created using the `mk_attr(...)` function and turns the foreground intensity and blink on.

Function `mk_token(...)`

Usage `token= mk_token(ch, attr);`
where
token is an int
ch is a char
attr is an unsigned char

Remarks This function receives an 8-bit character and an 8-bit attribute and returns a 16-bit token.

Function `mk_char_attr(...)`

Usage `mk_char_attr(token, *ch, *attr);`

where

token is an int

ch is a pointer to a char

attr is a pointer to an unsigned char

Remarks This function splits a 16-bit screen token and places the character in the char pointed to by *ch and the attribute in the unsigned char pointed to by *attr.

Demonstration of useful screen management routines

Figure 4-1 presents the source code listing to PROG4-1.C. This program demonstrates the use of many screen management functions.

4-1 The source code listing to PROG4-1.C.

```
////////////////////////////////////
//
// prog4-1.c
//
// Demonstrates:
//
//   scrn_init()
//   -----
//   Initializes the screen. It's
//   required only by DOS programs.
//   It's coded in the OS/2 FS library
//   for migration purposes only.
//
//   mk_attr()
//   -----
//   Makes a text attribute. This
//   attribute controls foreground
//   color, background color,
//   foreground intensity and
//   foreground blink.
//
//   scrn_change_attr()
//   -----
//   Changes the screen attributes
//   without altering characters.
//
//
//   scrn_write()
//   -----
//   Writes a string to the screen
//   at a designated row and column
//   location using a specified
//   screen attribute.
```

4-1 Continued.

```
//
//  scrn_repeat_char()
//  -----
//  Repeatedly writes a character
//  to the screen starting at a
//  designated row and column
//  location using a specified
//  attribute.
//
////////////////////////////////////
////////////////////////////////////
//
//  Compiler includes here
//
#include <stdio.h>

////////////////////////////////////
//
//  Library includes here
//

#include "tproto.h"

////////////////////////////////////
//
//  main function
//
void main()
{
    //
    //  initialize the screen
    //
    scrn_init();

    //
    //  save the current cursor location
    //
    cu_save_loc();

    //
    //  save the current cursor size
    //
    cu_save_size();

    //
    //  save the screen
    //
    scrn_save();

    //
    //  print message to the screen
    //
    scrn_repeat_char(0,
                     0,
                     80,
```



```

        ' ',
        mk_attr(WHITE, BLUE, ON_INTENSITY, OFF_BLINK));

scrn_write(0,
           0,
           0,
           "Press any key to alter screen attributes",
           mk_attr(WHITE, BLUE, ON_INTENSITY, OFF_BLINK));

//
// stop program and wait for key press
//

kb_read();

//
// alter screen attributes
//

scrn_change_attr(mk_attr(WHITE, BLUE, ON_INTENSITY, OFF_BLINK));

//
// print another message
//

scrn_write(1,
           0,
           0,
           "Press any key to restore original screen",
           mk_attr(WHITE, RED, ON_INTENSITY, ON_BLINK));

//
// stop program and wait for key press
//

kb_read();

//
// restore the screen
//

scrn_restore();

//
// restore cursor location
//

cu_rest_loc();

//
// restore previously saved cursor size
//

cu_rest_size();

}

//
////////////////////////////////////

```

4-1 Ends.

Introduction to menu bar / drop down window creation

Figure 4-2 presents the source code listing to PROG4-2.C. This program begins to explore one way to create a text based menu bar / drop down window user interface using the screen functions presented earlier.

4-2 The source code listing to PROG4-2.C.

```
////////////////////////////////////
//
// prog4-2.c
//
// Demonstrates:
//
//   scrn_clear()
//   -----
//   Clears the screen using the
//   Normal(7) attribute
//
//   scrn_attr()
//   -----
//   Alters a screen attribute at a
//   specified row and column
//   location.
//
//   setRect()
//   -----
//   Initialize rectangular structure
//
//   boxRect()
//   -----
//   Draw a box on the screen
//
//   saveRect()
//   -----
//   Save rectangular screen image
//
//   restRect()
//   -----
//   Restore rectangular screen image
//
//   dsyRect()
//   -----
//   Destroy rectangular structure
//
////////////////////////////////////

////////////////////////////////////
//
// Compiler includes here
//

#include <stdio.h>
#include <memory.h>

////////////////////////////////////
//
// Library includes here
//
```

```

#include "tproto.h"

////////////////////////////////////
//
// function prototypes
//

int    openDD_File(void);
int    openDD_Print(int num_copies);
int    openDD_Options(void);
int    openDD_Help(void);
void   print_menu_bar(void);

//
////////////////////////////////////
////////////////////////////////////
//
// print the menu bar
//

void print_menu_bar()
{
    UCHAR attr1, attr2;

    //
    // make the attributes
    //

    attr1= mk_attr(WHITE, BLUE, OFF_INTENSITY, OFF_BLINK);
    attr2= mk_attr(WHITE, BLUE, ON_INTENSITY, OFF_BLINK);

    //
    // print menu bar
    //

    scrn_attr(0, 0, 80, attr1);

    scrn_write(0,
               0,
               0,
               " File Print Options Help",
               attr1);

    //
    // highlight hot keys
    //

    scrn_attr(0, 1, 1, attr2);           // highlight F
    scrn_attr(0, 7, 1, attr2);           // highlight E
    scrn_attr(0, 14, 1, attr2);          // highlight O
    scrn_attr(0, 23, 1, attr2);          // highlight H

}

//
////////////////////////////////////

```

4-2 Continued.

```
////////////////////////////////////////
//
// open File drop down window
//

int openDD_File()
{
    UCHAR attr1, attr2, exit_flag= 0;
    RECT *R;
    int key, oldrow= 0, newrow= 0;

    //
    // set attributes
    //

    attr1= mk_attr(BLUE, WHITE, OFF_INTENSITY, OFF_BLINK);
    attr2= mk_attr(BLUE, WHITE, ON_INTENSITY, OFF_BLINK);

    //
    // initialize the rectangular structure
    //

    R= setRect(R,
               1,
               0,
               1 + 3,
               1 + 7);

    //
    // save the screen image under the rectangle
    //

    saveRect(R);

    //
    // draw a rectangular box under the File option
    // using a Single sided border (S_S_S_S);
    //

    boxRect(R,
            S_S_S_S,
            attr1);

    //
    // attributes change on File option
    //

    scrn_attr(0, 0, 6, attr1);

    //
    // main keyboard loop
    //

    do {

        // write the menu items to the screen

        scrn_write(2,
                   1,
                   0,
```

```

        " Open ",
        attr1);

scrn_write(3,
    1,
    0,
    " Exit ",
    attr1);

scrn_attr(oldrow + 2, 2, 5, attr1);

scrn_attr(2,
    2,
    1,
    attr2);

scrn_attr(3,
    3,
    1,
    attr2);

// draw highlight bar

scrn_attr(newrow + 2, 1, 7, mk_attr_inverse(attr1));

key= kb_read();

// process key press

switch(key) {

    // Item selected so
    // break from the loop

    case ENTER:
        if(newrow == 1) {
            key= ALT_X;
        }

        else {
            key= ALT_0;
        }

        exit_flag= 1;
        break;

    // select file for printing
    // quit program
    // no action

    case ALT_0:
    case ALT_X:
    case ESCAPE:
        exit_flag= 1;
        break;

    // move highlight bar down

    case DOWN_ARROW:
    case DOWN_ARROW_K:

```

4-2 Continued.

```
        if(newrow == 1) {
            newrow= 0;
        }

        else {
            newrow++;
        }
        break;

        // move highlight bar up

        case UP_ARROW:
        case UP_ARROW_K:
            if(newrow == 0) {
                newrow= 1;
            }

            else {
                newrow--;
            }
            break;
    }

    } while(!exit_flag);

    // restore original screen image
    restRect(R);

    // destroy the RECT structure (free memory)
    dsyRect(R);

    //
    // return menu bar to original state
    //

    print_menu_bar();

    //
    // return the key press
    //

    return key;
}

//
////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
//
// open Print drop down window
//

int openDD_Print(int num_copies)
{
    UCHAR attr1, attr2, exit_flag= 0;
    RECT *R;
    char  buffer[20];
```

```

int    key, oldrow= 0, newrow= 0;

    //
    // set attributes
    //

    attr1= mk_attr(BLUE, WHITE, OFF_INTENSITY, OFF_BLINK);
    attr2= mk_attr(BLUE, WHITE, ON_INTENSITY, OFF_BLINK);

    //
    // initialize the rectangular structure
    //

    R= setRect(R,
               1,
               6,
               1 + 3,
               6 + 22);

    //
    // save the screen image under the rectangle
    //

    saveRect(R);

    //
    // draw a rectangular box under the File option
    // using a Single sided border (S_S_S_S);
    //

    boxRect(R,
            S_S_S_S,
            attr1);

    //
    // attributes change on Print option
    //

    scrn_attr(0, 6, 7, attr1);

    //
    // main keyboard loop
    //

    do {

        // write the menu items to the screen

        scrn_write(2,
                   1 + 6,
                   0,
                   " Number of Copies      ",
                   attr1);

        memset(buffer, 0, 10);

        sprintf(buffer, "% 3d", num_copies);

        scrn_write(2,
                   1 + 6 + 18,
                   0,

```

4-2 Continued.

```
        buffer,
        attr1);

    scrn_write(3,
        1 + 6,
        0,
        " Start Print Job... ",
        attr1);

    scrn_attr(oldrow + 2, 2 + 6, 21, attr1);

    scrn_attr(2,
        2 + 6,
        1,
        attr2);

    scrn_attr(3,
        6 + 6,
        1,
        attr2);

    // draw highlight bar

    scrn_attr(newrow + 2, 1 + 6, 21, mk_attr_inverse(attr1));

    // get key press

    key= kb_read();

    // process key press

    switch(key) {

        // Item selected so
        // break from the loop

        case ENTER:
            if(newrow == 1) {
                key= ALT_X;
            }

            else {
                key= ALT_0;
            }

            exit_flag= 1;
            break;

        // select file for printing
        // quit program
        // no action

        case ALT_0:
        case ALT_X:
        case ESCAPE:
            exit_flag= 1;
            break;
```



```

        // move highlight bar down

        case DOWN_ARROW:
        case DOWN_ARROW_K:
            if(newrow == 1) {
                newrow= 0;
            }

            else {
                newrow++;
            }
            break;

        // move highlight bar up

        case UP_ARROW:
        case UP_ARROW_K:
            if(newrow == 0) {
                newrow= 1;
            }

            else {
                newrow--;
            }
            break;
    }

    } while(!exit_flag);

    // restore original screen image
    restRect(R);

    // destroy RECT structure
    dsyRect(R);

    //
    // return menu bar to original state
    //

    print_menu_bar();

    //
    // return the key press
    //

    return key;
}

//
////////////////////////////////////
//
////////////////////////////////////
//
// open Options drop down window
//

int openDD_Options()
{

```

4-2 Continued.

```
UCHAR attr1, attr2, exit_flag= 0;
RECT  *R;
int    key, oldrow= 0, newrow= 0;

    //
    // set attributes
    //

    attr1= mk_attr(BLUE, WHITE, OFF_INTENSITY, OFF_BLINK);
    attr2= mk_attr(BLUE, WHITE, ON_INTENSITY, OFF_BLINK);

    //
    // initialize the rectangular structure
    //

    R= setRect(R,
               1,
               13,
               1 + 3,
               13 + 24);

    //
    // save the screen image under the rectangle
    //

    saveRect(R);

    //
    // draw a rectangular box under the File option
    // using a Single sided border (S_S_S_S);
    //

    boxRect(R,
            S_S_S_S,
            attr1);

    //
    // attributes change on Options option
    //

    scrn_attr(0, 13, 9, attr1);

    //
    // main keyboard loop
    //

    do {

        // write the menu items to the screen

        scrn_write(2,
                   1 + 13,
                   0,
                   " Select Print Options  ",
                   attr1);

        scrn_write(3,
                   1 + 13,
                   0,
                   " Install Printer Codes  ",
```

```

        attr1);

scrn_attr(oldrow + 2, 2 + 13, 23, attr1);

scrn_attr(2,
          2 + 13,
          1,
          attr2);

scrn_attr(3,
          6 + 13,
          1,
          attr2);

// draw highlight bar
scrn_attr(newrow + 2, 1 + 13, 23, mk_attr_inverse(attr1));

// get key press
key= kb_read();

// process key press
switch(key) {
    // Item selected so
    // break from the loop
    case ENTER:
        if(newrow == 1) {
            key= ALT_X;
        }

        else {
            key= ALT_0;
        }
        exit_flag= 1;
        break;

    // select file for printing
    // quit program
    // no action

    case ALT_0:
    case ALT_X:
    case ESCAPE:
        exit_flag= 1;
        break;

    // move highlight bar down

    case DOWN_ARROW:
    case DOWN_ARROW_K:
        if(newrow == 1) {
            newrow= 0;
        }

        else {
            newrow++;

```

4-2 Continued.

```
        }
        break;

        // move highlight bar up

        case UP_ARROW:
        case UP_ARROW_K:
            if(newrow == 0) {
                newrow= 1;
            }

            else {
                newrow--;
            }
            break;
    }

    } while(!exit_flag);

    // restore original screen image
    restRect(R);

    // destroy RECT structure
    dsyRect(R);

    //
    // return menu bar to original state
    //

    print_menu_bar();

    //
    // return the key press
    //

    return key;
}

//
////////////////////////////////////
////////////////////////////////////
//
// open help drop down window
//

int openDD_Help()
{
    UCHAR attr1, attr2, exit_flag= 0;
    RECT *R;
    int key, oldrow= 0, newrow= 0;

    //
    // set attributes
    //
```

```

attr1= mk_attr(BLUE, WHITE, OFF_INTENSITY, OFF_BLINK);
attr2= mk_attr(BLUE, WHITE, ON_INTENSITY, OFF_BLINK);

//
// initialize the rectangular structure
//

R= setRect(R,
           1,
           22,
           1 + 3,
           22 + 19);

//
// save the screen image under the rectangle
//

saveRect(R);

//
// draw a rectangular box under the File option
// using a Single sided border (S_S_S_S);
//

boxRect(R,
        S_S_S_S,
        attr1);

//
// attributes change on Help option
//

scrn_attr(0, 22, 6, attr1);

//
// main keyboard loop
//

do {

    // write the menu items to the screen

    scrn_write(2,
               1 + 22,
               0,
               " Program Info... ",
               attr1);

    scrn_write(3,
               1 + 22,
               0,
               " Draneol Info... ",
               attr1);

    scrn_attr(oldrow + 2, 2 + 22, 5, attr1);

    scrn_attr(2,
               2 + 22,
               1,
               attr2);

```

4-2 Continued.

```
    scrn_attr(3,
              2 + 22,
              1,
              attr2);

    // draw highlight bar

    scrn_attr(newrow + 2, 1 + 22, 18, mk_attr_inverse(attr1));

    // get key press

    key= kb_read();

    // process key press

    switch(key) {

        // Item selected so
        // break from the loop

        case ENTER:
            if(newrow == 1) {
                key= ALT_P;
            }

            else {
                key= ALT_M;
            }
            exit_flag= 1;
            break;

        // select program help
        // select mentaur help
        // no action

        case ALT_P:
        case ALT_M:
        case ESCAPE:
            exit_flag= 1;
            break;

        // quit program

        case ALT_X:
            exit_flag= 1;
            break;

        // move highlight bar down

        case DOWN_ARROW:
        case DOWN_ARROW_K:
            if(newrow == 1) {
                newrow= 0;
            }

            else {
                newrow++;
            }
            break;
```

```

        // move highlight bar up

        case UP_ARROW:
        case UP_ARROW_K:
            if(newrow == 0) {
                newrow= 1;
            }

            else {
                newrow--;
            }
            break;
        }

    } while(!exit_flag);

    // restore original screen image
    restRect(R);

    // destroy RECT structure
    dsyRect(R);

    //
    // return menu bar to original state
    //

    print_menu_bar();

    //
    // return the key press
    //

    return key;
}

//
////////////////////////////////////
////////////////////////////////////
//
// main function
//

void main()
{
    UCHAR    attr2;
    UCHAR    exit_flag= 0;
    int      key, ret_val, num_print_copies= 1;

    //
    // initialize attribute
    //

    attr2= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);

    //
    // initialize the screen
    //

```

4-2 Continued.

```
scrn_init();

//
// save the current cursor location
//

cu_save_loc();

//
// save the current cursor size
//

cu_save_size();

//
// save the screen
//

scrn_save();

//
// turn off the cursor
//

cu_remove();

//
// clear the screen
//

scrn_clear();

//
// alter screen attributes
//

scrn_change_attr(attr2);

//
// print menu bar
//

print_menu_bar();

//
// print exit message at the screen bottom
// lower right
//

scrn_repeat_char(23,
                 0,
                 80,
                 196,
                 mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK));

scrn_write(24,
           58,
           0,
```



```

        "Leave Program: ALT-X",
        mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK));

//
// main program loop
//

do {
    // get the keyboard

    key= kb_read();

    // process key press

    switch(key) {

        case ALT_X:
            exit_flag= 1;
            break;

        case ALT_F:
            // open drop down window

            ret_val= openDD_File();

            if(ret_val == ALT_X) {
                exit_flag= 1;
            }
            break;

        case ALT_O:
            // open drop down window

            ret_val= openDD_Options();
            break;

        case ALT_P:
            // open drop down window

            ret_val= openDD_Print(num_print_copies);
            break;

        case ALT_H:
            // open drop down window

            ret_val= openDD_Help();
            break;
    }
} while(!exit_flag);

//
// restore the screen
//

scrn_restore();

//
// restore cursor location
//

```

4-2 Continued.

```
cu_rest_loc();

//
// restore previously saved cursor size
//

cu_rest_size();

//
// display the cursor
//

cu_display();

}

//
////////////////////////////////////
```

4-2 Ends.

Summary

This chapter presented many useful screen and screen attribute management functions. These included the ability to:

- Create a screen attribute
- Alter attributes in a variety of ways
- Write a character and attribute to the screen
- Alter screen attributes without changing the characters
- Write a string of characters to the screen
- Read a character and attribute from the screen

Chapter 5 introduces mouse management function descriptions and demonstration programs.

5

Mouse management demonstration programs

This chapter contains demonstration programs that show how to read text mouse cursor row and column location along with button press status. When the routines in this chapter are combined with those within the keyboard management arena all the foundation blocks are there to create a keyboard and mouse event handler.

Table 5-1 presents the function prototypes for the mouse related functions.

Table 5-1 Mouse function prototypes.

int	ms_init(void);
void	ms_on(void);
void	ms_off(void);
int	ms_status(int *x, int *y);
void	ms_map_display(int row, int col, int key_val);

Mouse function descriptions

Function ms_init()

Usage buttons= ms__init();
 where
 buttons is an int

Remarks On successful initialization of the mouse this function returns the number of buttons on the mouse. If function `ms__init(...)` returns `0xffff`, the mouse was not properly initialized.

Function `ms__on()`

Usage `ms__on();`

Remarks This function turns the mouse on (makes it visible).

Function `ms__off()`

Usage `ms__off();`

Remarks This function turns the mouse off (makes it invisible).

Function `ms__status(...)`

Usage `status= ms__status(*x, *y);`

where

`status` is an int

`x` is a pointer to an int

`y` is a pointer to an int

Remarks This function returns a 0 when there is no mouse button event. When there is a mouse button event a 1 (left button) or 2 (right button) is returned to `status`. Once the button is pressed, you can check the location of the mouse via `x` and `y` and determine the action which should be taken.

Function `ms__map_display(...)`

Usage `ms__map_display(row, col, esc__key__val);`

where

`row` is an int

`col` is an int

`esc__key__val` is an int

Remarks I wrote this function to help map areas of the screen. When you invoke this function, program execution is held within the mapping routine. When you press a mouse button, the row and column location of the mouse is printed to the screen at `row` and `col`. The `esc__key__val` is the key press value that terminates the mapping function and allows your program to continue executing. I repeatedly used this function when developing dialog boxes or drop down windows; it proves very useful.

Map your full screen OS/2 or DOS display using the mouse

Figure 5-1 presents the source code listing to PROG5-1.C. This program shows how to read mouse button press status along with the row and column location of the mouse.

5-1 The source code listing to PROG5-1.C.

```
////////////////////////////////////
//
// prog5-1.c
//
// Description:
//   This is a demonstration of a
//   mouse and keyboard event
//   handler that will help you
//   map your character display.
//   The debug-type event handler
//   was used in mapping the
//   menu bar/ drop window version
//   of the print utility presented
//   in Chapter 6.
//
//
//
//
// Compiler includes here
//
#include <stdlib.h>
#include <stdio.h>
#include <memory.h>
//
//
// Library includes here
//
#include "tproto.h"
//
//
// global variable
//
int    mouse_installed= 0;
//
//
// main function
//
void main()
{
```

5-1 Continued.

```
//
// initialize the screen
//

scrn_init();

//
// clear the screen
//

scrn_clear();

//
// move the cursor
//

cu_move(1, 0);

//
// remove the cursor from the display
//

cu_remove();

//
// check to see if the mouse is installed
//

mouse_installed= ms_init();

//
// if the mouse is not installed print the
// not installed message and return to the OS
//

if(!mouse_installed) {
    printf("Mouse not installed\n");
    exit(0);
}

//
// turn on the mouse
//

ms_on();

//
// map the mouse display to get dialog box coordinates
//

ms_map_display(10, 10, F10);

//
// turn off the mouse
//

ms_off();

//
// display the cursor
```

```

//
    cu_display();
}

//
////////////////////////////////////

```

5-1 Ends.

Map the user interface presented in PROG4-2

Figure 5-2 presents the source code listing to PROG5-2.C. This program demonstrates how to begin the mapping process for two drop down windows. In essence, it is part of the window mapping process. The process goes something like this:

- Write a program that displays the window.
- Invoke `ms__map__display(...)` function and simply move the mouse about the screen noting the row and column locations of the mouse where program action should be taken upon a button press.

5-2 The source code listing to PROG5-2.C.

```

////////////////////////////////////
//
// prog5-2.c
//
// Description:
//   Real world mapping demo
//   of PROG4-2.C
//

////////////////////////////////////
//
// Compiler includes here
//

#include <stdio.h>
#include <stdlib.h>
#include <memory.h>

////////////////////////////////////
//
// Library includes here
//

#include "tproto.h"

////////////////////////////////////
//
// function prototypes here
//

void print_menu_bar(void);
void print_print_DD(void);

```

5-2 Continued.

```
////////////////////////////////////
//
// global variable
//

int    mouse_installed= 0;

////////////////////////////////////
//
// main function
//

void main()
{

    //
    // initialize the screen
    //

    scrn_init();

    //
    // clear the screen
    //

    scrn_clear();

    //
    // move the cursor
    //

    cu_move(1, 0);

    //
    // remove the cursor from the display
    //

    cu_remove();

    //
    // initialize the mouse
    //

    mouse_installed= ms_init();

    //
    // if the mouse is not installed then
    // print the appropriate message and return to
    // the OS
    //

    if(!mouse_installed) {
        printf("Mouse not installed\n");
        exit(0);
    }

    //////////////////////////////////
    //
    // interface displayed here
```



```

//
print_menu_bar();

//
// turn on the mouse
//

ms_on();

//
// map the menu bar
//

ms_map_display(10, 10, F10);

//
// turn off the mouse
//

ms_off();

//
// clear the screen
//

scrn_clear();

//
// move the cursor
//

cu_move(0, 0);

//
// display the cursor
//

cu_display();
}

////////////////////////////////////
//
// Print the menu bar
//
// This function prints the area of the
// screen which is to be mapped by the
// mouse.
//

void print_menu_bar()
{
    UCHAR attr1, attr2;
    RECT *R, *R1;

    attr1= mk_attr(WHITE, BLUE, OFF_INTENSITY, OFF_BLINK);
    attr2= mk_attr(WHITE, BLUE, ON_INTENSITY, OFF_BLINK);

    scrn_attr(0, 0, 80, attr1);

```

5-2 Continued.

```
scrn_write(0,  
           0,  
           0,  
           " File Print Options Help",  
           attr1);
```

```
R= setRect(R,  
           1,  
           0,  
           1 + 3,  
           1 + 7);
```

```
boxRect(R,  
         S_S_S_S,  
         attr1);
```

```
scrn_write(2,  
           1,  
           0,  
           " Open ",  
           attr1);
```

```
scrn_write(3,  
           1,  
           0,  
           " Exit ",  
           attr1);
```

```
scrn_attr(2,  
          2,  
          1,  
          attr2);
```

```
scrn_attr(3,  
          3,  
          1,  
          attr2);
```

```
R1= setRect(R1,  
            1,  
            22,  
            1 + 3,  
            22 + 20);
```

```
boxRect(R1,  
         S_S_S_S,  
         attr1);
```

```
scrn_attr(0, 22, 6, attr1);
```

```
scrn_write(2,
```

```

        1 + 22,
        0,
        " Program Info... ",
        attr1);

    scrn_write(3,
        1 + 22,
        0,
        " Dranoel Info... ",
        attr1);


    scrn_attr(2,
        2 + 22,
        1,
        attr2);

    scrn_attr(3,
        2 + 22,
        1,
        attr2);

}

////////////////////////////////////
//
// Print the drop down window
//
// This function prints the area of the
// screen that is to be mapped by the
// mouse.
//

void print_print_DD()
{
    UCHAR attr1;
    RECT *R;

    attr1= mk_attr(BLUE, WHITE, OFF_INTENSITY, OFF_BLINK);

    R= setRect(R,
        1,
        6,
        1 + 3,
        6 + 22);

    boxRect(R,
        S_S_S_S,
        attr1);

    scrn_write(2,
        1 + 6,
        0,

```

5-2 Continued.

```
        " Number of Copies    ",
        attr1);

    scrn_write(3,
        1 + 6,
        0,
        " Start Print Job...  ",
        attr1);

}

//
////////////////////////////////////
```

5-2 Ends.

PROG4-2's interface becomes keyboard and mouse driven

Figure 5-3 presents the source code listing to PROG5-3.C. This shell-type program is one way to implement a keyboard and mouse driven menu bar / drop down window user interface.

5-3 The source code listing to PROG5-3.C.

```
////////////////////////////////////
//
// prog5-3.c
//
// Demonstrates:
//   Integration of mouse into user
//   interface.
//
////////////////////////////////////
//
////////////////////////////////////
//
// Compiler includes here
//

#include <stdio.h>
#include <memory.h>

////////////////////////////////////
//
// Library includes here
//

#include "tproto.h"

////////////////////////////////////
//
// function prototypes
//

int    openDD_File(void);
int    openDD_Print(int num_copies);
```

```

int    openDD_Options(void);
int    openDD_Help(void);
void   print_menu_bar(void);

//
////////////////////////////////////

////////////////////////////////////
//
// global data
//

int    mouse_installed;
char   FILE_ALT_O[]= {"FILE: Open           "};
char   FILE_ALT_X[]= {"FILE: Exit           "};
char   FILE_ALT_A[]= {"FILE: About          "};
char   PRINT_ALT_N[]= {"PRINT: Number of copies "};
char   PRINT_ALT_S[]= {"PRINT: Start print job "};
char   OPTIONS_ALT_P[]= {"OPTIONS: Select printer "};
char   OPTIONS_ALT_S[]= {"OPTIONS: Select print options "};
char   OPTIONS_ALT_I[]= {"OPTIONS: Install printer codes "};
char   HELP_ALT_P[]= {"HELP: Program help "};
char   HELP_ALT_M[]= {"HELP: Dranoel help "};
char   ERASE_message[]= {" "};

//
////////////////////////////////////

////////////////////////////////////
//
// print the menu bar
//

void print_menu_bar()
{
    UCHAR attr1, attr2;

    //
    // make the attributes
    //

    attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);
    attr2= mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK);

    //
    // print menu bar
    //

    scrn_attr(0, 0, 80, attr1);

    scrn_write(0,
               0,
               0,
               " File Print Options Help",
               attr1);

    //
    // highlight hot keys
    //

```

5-3 Continued.

```
    scrn_attr(0, 1, 1, attr2);          // highlight F
    scrn_attr(0, 7, 1, attr2);          // highlight E
    scrn_attr(0, 14, 1, attr2);         // highlight O
    scrn_attr(0, 23, 1, attr2);         // highlight H
}

//
////////////////////////////////////
////////////////////////////////////
//
//
// open File drop down window
//

int openDD_File()
{
    UCHAR attr1, attr2, exit_flag= 0;
    char split_items[12]= { 195, 196, 196, 196, 196, 196,
                           196, 196, 196, 196, 196, 180 };
    RECT *R;
    int key, oldrow= 0, newrow= 0;
    int x, y;

    //
    // set attributes
    //

    attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);
    attr2= mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK);

    //
    // initialize the rectangular structure
    //

    R= setRect(R,
               1,
               0,
               1 + 5,
               1 + 10);

    //
    // save the screen image under the rectangle
    //

    saveRect(R);

    //
    // draw a rectangular box under the File option
    // using a Single sided border (S_S_S_S);
    //

    boxRect(R,
            S_S_S_S,
            attr1);

    //
    // attributes change on File option
    //
```

```

scrn_attr(0, 0, 6, attr1);

// write the menu items to the screen

scrn_write(2,
           1,
           0,
           " Open      ",
           attr1);

scrn_write(3,
           1,
           0,
           " Exit      ",
           attr1);

scrn_write(4,
           0,
           12,
           split_items,
           attr1);

scrn_write(5,
           1,
           0,
           " About... ",
           attr1);

scrn_attr(oldrow + 2, 2, 8, attr1);

scrn_attr(2,
          2,
          1,
          attr2);

scrn_attr(3,
          3,
          1,
          attr2);

scrn_attr(5,
          2,
          1,
          attr2);

// draw highlight bar
scrn_attr(newrow + 2, 1, 10, mk_attr_inverse(attr1));

if(mouse_installed) {
    ms_on();
}
//
// main keyboard loop
//

do {
    if(oldrow != newrow) {
        if(mouse_installed) {

```

5-3 Continued.

```
        ms_off();
    }

    // write the menu items to the screen

    scrn_write(2,
               1,
               0,
               " Open      ",
               attr1);

    scrn_write(3,
               1,
               0,
               " Exit      ",
               attr1);

    scrn_write(4,
               0,
               12,
               split_items,
               attr1);

    scrn_write(5,
               1,
               0,
               " About... ",
               attr1);

    scrn_attr(oldrow + 2, 2, 8, attr1);

    scrn_attr(2,
              2,
              1,
              attr2);

    scrn_attr(3,
              3,
              1,
              attr2);

    scrn_attr(5,
              2,
              1,
              attr2);

    // draw highlight bar

    scrn_attr(newrow + 2, 1, 10, mk_attr_inverse(attr1));

    oldrow= newrow;

    // if mouse installed turn on the mouse

    if(mouse_installed) {
        ms_on();
    }

}
```



```

key= kb_status();

if((!key) && (mouse_installed)) {

    key= ms_status(&x, &y);

    if((key == 1) && (x >= 8) && (x <= (48 + 32)) && (y == 16)) {
        key= ALT_0;
    }

    if((key == 1) && (x >= 8) && (x <= (48 + 32)) && (y == 24)) {
        key= ALT_X;
    }

    if((key == 1) && (x >= 8) && (x <= (48 + 32)) && (y == 40)) {
        key= ALT_A;
    }

    if((key == 1) && (y >= 8)) {
        key= ESCAPE;
    }

    if((key == 2) && (y >= 8)) {
        key= RIGHT_ARROW;
    }
}

// process key press

switch(key) {

    // Item selected so
    // break from the loop

    case ENTER:
        if(newrow == 1) {
            key= ALT_X;
        }

        else {
            key= ALT_0;
        }
        exit_flag= 1;
        break;

    // select file for printing
    // quit program
    // no action

    case RIGHT_ARROW:
    case RIGHT_ARROW_K:
    case LEFT_ARROW:
    case LEFT_ARROW_K:
    case ALT_A:
    case ALT_0:
    case ALT_X:
    case ESCAPE:
        exit_flag= 1;
        break;

    // move highlight bar down

```

5-3 Continued.

```
        case DOWN_ARROW:
        case DOWN_ARROW_K:
            if(newrow == 1) {
                newrow= 3;
            }

            else if(newrow == 3) {
                newrow= 0;
            }

            else {
                newrow++;
            }
            break;

        // move highlight bar up

        case UP_ARROW:
        case UP_ARROW_K:
            if(newrow == 0) {
                newrow= 3;
            }

            else if(newrow == 3) {
                newrow= 1;
            }

            else {
                newrow--;
            }
            break;
    }

    } while(!exit_flag);

if(mouse_installed) {
    ms_off();
}

restRect(R);
dsyRect(R);

//
// return menu bar to original state
//

print_menu_bar();

if(mouse_installed) {
    ms_on();
}

//
// return the key press
//

return key;
```

```

}

//
////////////////////////////////////

////////////////////////////////////
//
// open Print drop down window
//

int openDD_Print(int num_copies)
{
    UCHAR attr1, attr2, exit_flag= 0;
    RECT  *R;
    char  buffer[20];
    int   key, oldrow= 0, newrow= 0;
    int   x, y;

    //
    // set attributes
    //

    attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);
    attr2= mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK);

    //
    // initialize the rectangular structure
    //

    R= setRect(R,
               1,
               6,
               1 + 3,
               6 + 22);

    //
    // save the screen image under the rectangle
    //

    saveRect(R);

    //
    // draw a rectangular box under the File option
    // using a Single sided border (S_S_S_S);
    //

    boxRect(R,
            S_S_S_S,
            attr1);

    //
    // attributes change on Print option
    //

    scrn_attr(0, 6, 7, attr1);

    // write the menu items to the screen

    scrn_write(2,
               1 + 6,
               0,

```

5-3 Continued.

```
        " Number of Copies    ",
        attr1);

memset(buffer, 0, 10);

sprintf(buffer, "% 3d", num_copies);

scrn_write(2,
           1 + 6 + 18,
           0,
           buffer,
           attr1);

scrn_write(3,
           1 + 6,
           0,
           " Start Print Job... ",
           attr1);

scrn_attr(oldrow + 2, 2 + 6, 21, attr1);

scrn_attr(2,
           2 + 6,
           1,
           attr2);

scrn_attr(3,
           2 + 6,
           1,
           attr2);

// draw highlight bar

scrn_attr(newrow + 2, 1 + 6, 21, mk_attr_inverse(attr1));

if(mouse_installed) {
    ms_on();
}

//
//  main keyboard loop
//

do {
    if(oldrow != newrow) {

        if(mouse_installed) {
            ms_off();
        }

        // write the menu items to the screen

        scrn_write(2,
                   1 + 6,
                   0,
                   " Number of Copies    ",
                   attr1);
```

```

memset(buffer, 0, 10);

sprintf(buffer,"% 3d", num_copies);

scrn_write(2,
            1 + 6 + 18,
            0,
            buffer,
            attr1);

scrn_write(3,
            1 + 6,
            0,
            "Start Print Job... ",
            attr1);

scrn_attr(oldrow + 2, 2 + 6, 21, attr1);

scrn_attr(2,
            2 + 6,
            1,
            attr2);

scrn_attr(3,
            2 + 6,
            1,
            attr2);

// draw highlight bar

scrn_attr(newrow + 2, 1 + 6, 21, mk_attr_inverse(attr1));

oldrow= newrow;

if(mouse_installed) {
    ms_on();
}

}

key= kb_status();

if((!key) && (mouse_installed)) {

    key= ms_status(&x, &y);

    if((key == 1) && (x >= 56) && (x <= (56 + (21 * 7)))
        && (y == 16)) {
        key= ALT_N;
    }

    if((key == 1) && (x >= 56) && (x <= (56 + (21 * 7)))
        && (y == 24)) {
        key= ALT_S;
    }

    if((key == 1) && (y >= 8)) {
        key= ESCAPE;
    }
}

```

5-3 Continued.

```
        if((key == 2) && (y >= 8)) {
            key= RIGHT_ARROW;
        }

    }

    // process key press

    switch(key) {

        // Item selected so
        // break from the loop

        case ENTER:
            if(newrow == 1) {
                key= ALT_X;
            }

            else {
                key= ALT_O;
            }
            exit_flag= 1;
            break;

        // select file for printing
        // quit program
        // no action

        case RIGHT_ARROW:
        case RIGHT_ARROW_K:
        case LEFT_ARROW:
        case LEFT_ARROW_K:
        case ALT_N:
        case ALT_S:
        case ESCAPE:
            exit_flag= 1;
            break;

        // move highlight bar down

        case DOWN_ARROW:
        case DOWN_ARROW_K:
            if(newrow == 1) {
                newrow= 0;
            }

            else {
                newrow++;
            }
            break;

        // move highlight bar up

        case UP_ARROW:
        case UP_ARROW_K:
            if(newrow == 0) {
                newrow= 1;
            }
    }
```

```

        else {
            newrow--;
        }
        break;
    }

    } while(!exit_flag);

//
// if the mouse is installed turn the mouse off
//

if(mouse_installed) {
    ms_off();
}

restRect(R);
dsyRect(R);

//
// return menu bar to original state
//

print_menu_bar();

if(mouse_installed) {
    ms_on();
}

//
// return the key press
//

return key;
}

//
////////////////////////////////////
////////////////////////////////////
//
// open Options drop down window
//

int openDD_Options()
{
    UCHAR attr1, attr2, exit_flag= 0;
    RECT *R;
    int key, oldrow= 0, newrow= 0;
    int x, y;

    //
    // set attributes
    //

    attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);
    attr2= mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK);

```

5-3 Continued.

```
//  
// initialize the rectangular structure  
//  
R= setRect(R,  
           1,  
           13,  
           1 + 4,  
           13 + 24);  
  
//  
// save the screen image under the rectangle  
//  
saveRect(R);  
  
//  
// draw a rectangular box under the File option  
// using a Single sided border (S_S_S_S);  
//  
boxRect(R,  
        S_S_S_S,  
        attr1);  
  
//  
// attributes change on Options option  
//  
scrn_attr(0, 13, 9, attr1);  
  
// write the menu items to the screen  
  
scrn_write(2,  
           1 + 13,  
           0,  
           " Select Printer  LPT1  ",  
           attr1);  
  
scrn_write(3,  
           1 + 13,  
           0,  
           " Select Print Options  ",  
           attr1);  
  
scrn_write(4,  
           1 + 13,  
           0,  
           " Install Printer Codes ",  
           attr1);  
  
scrn_attr(oldrow + 2, 2 + 13, 23, attr1);  
  
scrn_attr(2,  
          2 + 13 + 7,  
          1,  
          attr2);
```



```

scrn_attr(3,
          2 + 13,
          1,
          attr2);

scrn_attr(4,
          2 + 13,
          1,
          attr2);

// draw highlight bar
scrn_attr(newrow + 2, 1 + 13, 23, mk_attr_inverse(attr1));

if(mouse_installed) {
    ms_on();
}

//
// main keyboard loop
//
do {
    if(oldrow != newrow) {
        if(mouse_installed) {
            ms_off();
        }

        // write the menu items to the screen

        scrn_write(2,
                   1 + 13,
                   0,
                   " Select Printer  LPT1  ",
                   attr1);

        scrn_write(3,
                   1 + 13,
                   0,
                   " Select Print Options  ",
                   attr1);

        scrn_write(4,
                   1 + 13,
                   0,
                   " Install Printer Codes ",
                   attr1);

        scrn_attr(oldrow + 2, 2 + 13, 23, attr1);

        scrn_attr(2,
                  2 + 13 + 7,
                  1,
                  attr2);

        scrn_attr(3,
                  2 + 13,

```

5-3 Continued.

```
        1,
        attr2);

    scrn_attr(4,
        2 + 13,
        1,
        attr2);

    // draw highlight bar

    oldrow= newrow;

    scrn_attr(newrow + 2, 1 + 13, 23, mk_attr_inverse(attr1));

    if(mouse_installed) {
        ms_on();
    }
}

key= kb_status();

if((!key) && (mouse_installed)) {

    key= ms_status(&x, &y);

    if((key == 1) && (x >= 112) && (x <= (112 + (24 * 8)))
        && (y == 16)) {
        key= ALT_P;
    }

    if((key == 1) && (x >= 112) && (x <= (112 + (24 * 8)))
        && (y == 24)) {
        key= ALT_S;
    }

    if((key == 1) && (x >= 112) && (x <= (112 + (21 * 7)))
        && (y == 32)) {
        key= ALT_I;
    }

    if((key == 1) && (y >= 8)) {
        key= ESCAPE;
    }

    if((key == 2) && (y >= 8)) {
        key= RIGHT_ARROW;
    }
}

// process key press

switch(key) {

    // Item selected so
    // break from the loop

    case ENTER:
        if(newrow == 1) {
            key= ALT_S;
        }
    }
}
```

```

        }

        else if(newrow == 2) {
            key= ALT_I;
        }

        else {
            key= ALT_P;
        }
        exit_flag= 1;
        break;

// select file for printing
// quit program
// no action

case RIGHT_ARROW:
case RIGHT_ARROW_K:
case LEFT_ARROW:
case LEFT_ARROW_K:
case ALT_P:
case ALT_S:
case ALT_I:
case ESCAPE:
    exit_flag= 1;
    break;

// move highlight bar down

case DOWN_ARROW:
case DOWN_ARROW_K:
    if(newrow == 2) {
        newrow= 0;
    }

    else {
        newrow++;
    }
    break;

// move highlight bar up

case UP_ARROW:
case UP_ARROW_K:
    if(newrow == 0) {
        newrow= 2;
    }
    else {
        newrow--;
    }
    break;
}

} while(!exit_flag);

//
// if the mouse is installed turn the mouse off
//

if(mouse_installed) {
    ms_off();
}

```

5-3 Continued.

```
    }

    restRect(R);

    dsyRect(R);

    //
    // return menu bar to original state
    //

    print_menu_bar();

    if(mouse_installed) {
        ms_on();
    }
    //
    // return the key press
    //

    return key;
}

//
////////////////////////////////////
////////////////////////////////////
//
// open help drop down window
//

int openDD_Help()
{
    UCHAR attr1, attr2, exit_flag= 0;
    RECT *R;
    int key, oldrow= 0, newrow= 0;
    int x, y;

    //
    // set attributes
    //

    attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);
    attr2= mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK);

    //
    // initialize the rectangular structure
    //

    R= setRect(R,
               1,
               22,
               1 + 3,
               22 + 19);

    //
    // save the screen image under the rectangle
    //

    saveRect(R);
```

```

//
// draw a rectangular box under the File option
// using a Single sided border (S_S_S_S);
//

boxRect(R,
        S_S_S_S,
        attr1);

//
// attributes change on Help option
//

scrn_attr(0, 22, 6, attr1);

// write the menu items to the screen

scrn_write(2,
           1 + 22,
           0,
           " Program Help... ",
           attr1);

scrn_write(3,
           1 + 22,
           0,
           " Draneol Help... ",
           attr1);

scrn_attr(oldrow + 2, 2 + 22, 5, attr1);

scrn_attr(2,
          2 + 22,
          1,
          attr2);

scrn_attr(3,
          2 + 22,
          1,
          attr2);

// draw highlight bar

scrn_attr(newrow + 2, 1 + 22, 18, mk_attr_inverse(attr1));

if(mouse_installed) {
    ms_on();
}

//
// main keyboard loop
//

do {

    if(oldrow != newrow) {

        if(mouse_installed) {
            ms_off();
        }
    }
}

```

5-3 Continued.

```
// write the menu items to the screen

scrn_write(2,
           1 + 22,
           0,
           " Program Info... ",
           attr1);

scrn_write(3,
           1 + 22,
           0,
           " Draneol Info... ",
           attr1);

scrn_attr(oldrow + 2, 2 + 22, 5, attr1);

scrn_attr(2,
          2 + 22,
          1,
          attr2);

scrn_attr(3,
          2 + 22,
          1,
          attr2);

// draw highlight bar

scrn_attr(newrow + 2, 1 + 22, 18, mk_attr_inverse(attr1));

oldrow= newrow;

if(mouse_installed) {
    ms_on();
}

key= kb_status();

if((!key) && (mouse_installed)) {
    key= ms_status(&x, &y);

    if((key == 1) && (x >= 184) && (x <= 328)
        && (y == 16)) {
        key= ALT_P;
    }

    if((key == 1) && (x >= 184) && (x <= 328)
        && (y == 24)) {
        key= ALT_M;
    }

    if((key == 1) && (y >= 8)) {
        key= ESCAPE;
    }

    if((key == 2) && (y >= 8)) {
        key= RIGHT_ARROW;
    }
}
```

```

    }
}

// process key press
switch(key) {

    // Item selected so
    // break from the loop

    case ENTER:
        if(newrow == 1) {
            key= ALT_P;
        }

        else {
            key= ALT_M;
        }
        exit_flag= 1;
        break;

    // select program help
    // select mentaur help
    // no action

    case RIGHT_ARROW:
    case RIGHT_ARROW_K:
    case LEFT_ARROW:
    case LEFT_ARROW_K:
    case ALT_P:
    case ALT_M:
    case ESCAPE:
        exit_flag= 1;
        break;

    // quit program

    case ALT_X:
        exit_flag= 1;
        break;

    // move highlight bar down

    case DOWN_ARROW_K:
    case DOWN_ARROW:
        if(newrow == 1) {
            newrow= 0;
        }

        else {
            newrow++;
        }
        break;

    // move highlight bar up

    case UP_ARROW:
    case UP_ARROW_K:
        if(newrow == 0) {

```

5-3 Continued.

```
        newrow= 1;
    }

    else {
        newrow--;
    }
    break;
}

} while(!exit_flag);

//
// if the mouse is installed then turn it off
//

if(mouse_installed) {
    ms_off();
}

restRect(R);
dsyRect(R);

if(mouse_installed) {
    ms_on();
}
//
// return menu bar to original state
//

print_menu_bar();

//
// return the key press
//

return key;
}

//
//
//
//
//
// main function
//

void main()
{
    UCHAR    attr2;
    UCHAR    exit_flag= 0;
    int      key, ret_val, num_print_copies= 1;
    int      x, y, counter;

    //
    // initialize the screen
    //
```



```

scrn_init();

//
// initialize the mouse
//

mouse_installed= ms_init();

//
// initialize attributes
//

attr2= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);

//
// save the current cursor location
//

cu_save_loc();

//
// save the current cursor size
//

cu_save_size();

//
// save the screen
//

scrn_save();

//
// turn off the cursor
//

cu_remove();

//
// clear the screen
//

scrn_clear();

//
// alter screen attributes
//

scrn_change_attr(attr2);

//
// print menu bar
//

print_menu_bar();

//
// fill in block characters
//

```

5-3 Continued.

```
for(counter= 1; counter < 24; counter++) {
    scrn_repeat_char(counter,
                      0,
                      80,
                      177,
                      mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK));
}

//
// print exit message at the screen bottom
// lower right
//

scrn_write(24,
           58,
           0,
           "Leave Program: ALT-X",
           mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK));

//
// turn on the mouse if installed
//

if(mouse_installed) {
    ms_on();
}

//
// main program loop
//

do {
    // get the keyboard - no wait for key press

    key= kb_status();

    // if there is no key press and the
    // mouse is installed then

    if(!key) && (mouse_installed)) {
        // check the status of the mouse

        key= ms_status( &x, &y);

        // if there is a left button press and
        // the mouse is at the top of the screen
        // then (using the data that you obtained
        // from PROG5-2) map the mouse location
        // to the appropriate key press

        if((key == 1) && (y == 0)) {
            if((x >= 0) && (x <= 40)) {
                key= ALT_F;
            }

            if((x >= 48) && (x <= 96)) {
                key= ALT_P;
            }
        }
    }
}
```

```

    }

    if((x >= 104) && (x <= 168)) {
        key= ALT_0;
    }

    if((x >= 176) && (x <= 216)) {
        key= ALT_H;
    }
}

////////////////////////
// check to see if ALX-X

if((key == 1) && (y == 192) && (x >= 456) && (x <= 632)) {
    key= ALT_X;
}
}

// process key press

new_window:

switch(key) {

    case ALT_X:
        exit_flag= 1;
        break;

    case ALT_F:
        if(mouse_installed) {
            ms_off();
        }

        scrn_write(24, 0, 0, ERASE_message, attr2);

        ret_val= openDD_File();

        if((ret_val == RIGHT_ARROW) || (ret_val == RIGHT_ARROW_K)) {
            key = ALT_P;
            goto new_window;
        }

        else if((ret_val == LEFT_ARROW) || (ret_val == LEFT_ARROW_K)) {
            key = ALT_H;
            goto new_window;
        }

        else if(ret_val == ALT_0) {
            scrn_write(24, 0, 0, FILE_ALT_0, attr2);
        }

        else if(ret_val == ALT_X) {
            scrn_write(24, 0, 0, FILE_ALT_X, attr2);
        }

        else {
            scrn_write(24, 0, 0, FILE_ALT_A, attr2);
        }
}

```

5-3 Continued.

```
        if(mouse_installed) {
            ms_on();
        }

        if(ret_val == ALT_X) {
            exit_flag= 1;
        }
        break;

case ALT_P:
    if(mouse_installed) {
        ms_off();
    }

    scrn_write(24, 0, 0, ERASE_message, attr2);

    ret_val= openDD_Print(num_print_copies);

    if((ret_val == RIGHT_ARROW) || (ret_val == RIGHT_ARROW_K)) {
        key = ALT_O;
        goto new_window;
    }

    else if((ret_val == LEFT_ARROW) || (ret_val == LEFT_ARROW_K)) {
        key = ALT_F;
        goto new_window;
    }

    else if(ret_val == ALT_N) {
        scrn_write(24, 0, 0, PRINT_ALT_N, attr2);
    }

    else {
        scrn_write(24, 0, 0, PRINT_ALT_S, attr2);
    }

    if(mouse_installed) {
        ms_on();
    }
    break;

case ALT_0:
    if(mouse_installed) {
        ms_off();
    }

    scrn_write(24, 0, 0, ERASE_message, attr2);

    ret_val= openDD_Options();

    if((ret_val == RIGHT_ARROW) || (ret_val == RIGHT_ARROW_K)) {
        key = ALT_H;
        goto new_window;
    }

    else if((ret_val == LEFT_ARROW) || (ret_val == LEFT_ARROW_K)) {
        key = ALT_P;
        goto new_window;
    }
```

```

    }

    else if(ret_val == ALT_P) {
        scrn_write(24, 0, 0, OPTIONS_ALT_P, attr2);
    }

    else if(ret_val == ALT_S) {
        scrn_write(24, 0, 0, OPTIONS_ALT_S, attr2);
    }

    else {
        scrn_write(24, 0, 0, OPTIONS_ALT_I, attr2);
    }

    if(mouse_installed) {
        ms_on();
    }
    break;

case ALT_H:
    if(mouse_installed) {
        ms_off();
    }

    scrn_write(24, 0, 0, ERASE_message, attr2);

    ret_val= openDD_Help();

    if((ret_val == RIGHT_ARROW) || (ret_val == RIGHT_ARROW_K)) {
        key = ALT_F;
        goto new_window;
    }

    else if((ret_val == LEFT_ARROW) || (ret_val == LEFT_ARROW_K)) {
        key = ALT_0;
        goto new_window;
    }

    else if(ret_val == ALT_P) {
        scrn_write(24, 0, 0, HELP_ALT_P, attr2);
    }

    else {
        scrn_write(24, 0, 0, HELP_ALT_M, attr2);
    }

    if(mouse_installed) {
        ms_on();
    }
    break;

}

} while(!exit_flag);

//
// turn off the mouse
//

```

5-3 Continued.

```
    if(mouse_installed) {
        ms_off();
    }

    //
    // restore the screen
    //

    scrn_restore();

    //
    // restore cursor location
    //

    cu_rest_loc();

    //
    // restore previously saved cursor size
    //

    cu_rest_size();

    //
    // display the cursor
    //

    cu_display();

}

//
////////////////////////////////////
```

5-3 Ends.

Summary

This chapter presented many useful mouse management functions. These functions allow you to:

- Initialize the mouse
- Examine the mouse for a button press event
- Examine the mouse for its row and column location
- Turn on the mouse (display it)
- Turn off the mouse (remove it from the display)
- Map the screen for mouse and text location

Chapter 6 introduces powerful character-based window management functions and useful demonstration programs.

6

Character-based window management demonstration programs

This chapter contains foundation character-based window management functions that take advantage of using a local coordinate system. The local coordinate system permits the programmer to conceive of the window as a mini-screen. The row 0 and column 0 location of the window with its local coordinate system is the upper left hand corner of the window border.

There are high-level and low-level window functions. Because the high-level window functions greatly ease the task of character-based window generation, I suggest that you stick with using them and avoid using the low-level routines.

Notice that many of the high-level window functions will have a familiar ring to them. For example, you'll note that function `wind__write(...)` is directly related to function `scrn__write(...)` and function `wind__char(...)` is directly related to function `scrn__char(...)`, etc...

We've carefully considered the naming conventions used for these functions. I believe that the migration library presented in this book will prove easy to use because many function names are differentiated only by their prefix. That way, once you get the hang of writing to the screen using a global coordinate system, you'll easily be able to translate that knowledge to writing to the window using the local coordinate system.

The method of popping up a window and then removing it and restoring the screen is quite simple. Here it is:

1. Save screen image where window will appear
2. Write window to screen

3. Operate in window
4. Save window image to memory
5. Restore previously saved screen image from

When you construct a window, you can choose among three border styles supported by the library. They are

Border ID	Meaning
S_S_S_S	Single line border all around
D_D_D_D	Double line border all around
D_D_S_S	Double line top and bottom border and single line left and right border

Note that the high-level window management functions are based on a foundation of screen and make management functions. Table 6-1 presents the function prototypes for the screen.

Table 6-1 High level window function prototypes.

WIND	*wind_init(WIND	*W_PTR,
	int	ulr,
	int	ulc,
	int	lrr,
	int	lrc,
	UCHAR	attr,
	int	border,
	char	*title);
int	wind_kb_edit(WIND	*W,
	char	*response,
	int	row,
	int	column,
	int	dlen,
	int	opt,
	UCHAR	attr);
void	wind_display(WIND	*W);
void	wind_remove(WIND	*W);
void	wind_attr(WIND	*W,
	int	row,
	int	col,
	int	length,
	UCHAR	attr);
void	wind_write(WIND	*W,
	int	row,
	int	col,
	int	length,
	char	*string,
	UCHAR	attr);
void	wind_char(WIND	*W,

Table 6-1 Continued.

		int	row,
		int	col,
		char	ch,
		UCHAR	attr);
void	wind_repeat_char(WIND	*W,	
		int	row,
		int	col,
		int	len,
		char	ch,
		UCHAR	attr);
void	wind_destroy(WIND	*W);	
int	wind_read_char(WIND	*W, int row, int col);	
void	wind_clear(void);		
void	wind_cu_move(WIND	*W, int row, int col);	

Window function descriptions

Function `wind__init(...)`

Usage `W= wind__init(W,`
 `ulr,`
 `ulc,`
 `lrr,`
 `lrc,`
 `attr,`
 `border,`
 `*title);`

where

`W` is a pointer to a window structure

`ulr` is an int

`ulc` is an int

`lrr` is an int

`lrc` is an int

`attr` is an unsigned char

`border` is an int

`title` is a pointer to a character string

Remarks This function is the fundamental window creation routine. It initializes the window structure, saves the screen image under the window and then writes the window. Variables `ulr` and `ulc` specify the upper left hand corner of the window. Variables `lrr` and `lrc` specify the lower right row and column of the window. Variable `attr` is the window attribute. Variable `border` designates the window border style. Variable `title` points to the window title string.

Function `wind__kb__edit(...)`

Usage `key= wind__kb__edit(W,
 response,
 row,
 column,
 dlen,
 opt,
 attr);`

where
key is an int
W is a pointer to a window structure
response is a pointer to a character buffer
row is an int
column is an int
dlen is an int
opt is an int
attr is an unsigned char

Remarks This function retrieves an alpha-numeric string from the keyboard. Variable `key` holds the terminating 16-bit key press value (Enter or Esc) and is used to take appropriate action on function return. Variable `response` points to a buffer that will contain the alpha-numeric string. Variables `row` and `column` set the entry string row and column location. Variable `opt` forces all caps (UPPER), all lowercase (LOWER) or upper- and lowercase (UPPER_LOWER). Variable `attr` is the entry field attribute.

Function `wind__display(...)`

Usage `wind__display(W);`

where
w is a pointer to a window structure

Remarks This function displays a window after it has been previously initialized and removed.

Function `wind__remove(...)`

Usage `wind__remove(W);`

where
W is a pointer to a window structure

Remarks This function removes a displayed window from the screen and restore the original screen image under the window.

Function `wind_attr(...)`

Usage `wind_attr(W,
 row,
 col,
 length,
 attr);`

where

`W` is a pointer to a window

`row` is an int

`col` is an int

`length` is an int

`attr` is an unsigned char

Remarks This function allows you to control the attributes within the window. It will prove extremely useful for highlighting areas of text within the window. Variables `row` and `col` point to the start area of the attribute change. Variable `length` determines the number of attributes to change and variable `attr` is the new window attribute. There is not any protection in this function that prevents window border overwrites. It is up to you to make sure that a window border is not overwritten.

Function `wind_write(...)`

Usage `wind_write(W,
 row,
 col,
 length,
 string,
 attr);`

where

`W` is a pointer to a window structure

`row` is an int

`col` is an int

`length` is an int

`string` points to a character buffer

`attr` is an unsigned int

Remarks This function writes a string to the window of a specified length at a designated row and column location. The string write attribute may also be designated. This function is identical to the `scrn_write(...)` function with the exception that it receives a pointer to a window structure as the first parameter and uses the local coordinate system of the

window for an origin. There is not any protection in this function which prevents window border overwrites. It is up to you to make sure that a window border is not overwritten.

Function `wind__char(...)`

Usage `wind__char(W,`
 `row,`
 `col,`
 `ch,`
 `attr);`

where

`W` is a pointer to a window structure

`row` is an int

`col` is an int

`attr` is an unsigned char

Remarks This function writes a character and attribute to a window at a specified row and column location using a designated attribute. This function is identical to the `scrn__char(...)` function with the exception that it receives a pointer to a window structure as the first parameter and uses the local coordinate system of the window for an origin.

Function `wind__repeat__char(...)`

Usage `wind__repeat__char(W,`
 `row,`
 `col,`
 `len,`
 `ch,`
 `attr);`

where

`W` is a pointer to a window structure

`row` is an int

`col` is an int

`len` is an int

`ch` is a char

`attr` is an unsigned char

Remarks This function writes a character to a window a specified number of times. There is no boundary checking in this function. It is up to you to make sure that there is not a situation where you will overwrite a window border.

Function `wind__destroy(...)`

Usage `wind__destroy(W);`
 where
 `W` is a pointer to a window structure

Remarks This function destroys a window structure by freeing up all memory that had been dynamically allocated during `wind__init(...)` function execution. Once this function has been invoked, the window structure must be reinitialized again.

Function `wind__read__char(...)`

Usage `token= wind__read__char(W, row, col);`
 where
 `token` is an int
 `W` is a pointer to a window structure
 `row` is an int
 `col` is an int

Remarks This function returns the character and attribute at a specified row and column location within a window.

Function `wind__cu__move(...)`

Usage `wind__cu__move(W, row, col);`
 where
 `W` is a pointer to a window structure
 `row` is an int
 `col` is an int

Remarks This function moves the cursor to a specified row and column window location.

Mapping a dialog box

The process of creating a dialog box, although not fully automated, proves quite simple nonetheless. It consists of three simple steps:

1. Use window functions to create your dialog box
2. Use function `ms__map__display(...)` to discover the mouse positions related to dialog box function. (It's easier to do than to describe!)
3. Modify the code in `PROG6-nnn`.

Figure 6-1 presents the source code listing to `PROG6-1.C`. This demonstration program shows how to map a dialog box so the mouse read locations can easily be calculated.

6-1 The source code listing to PROG6-1.C.

```
////////////////////////////////////
//
// prog6-1.c
//
// Demonstrates:
//   Mapping a dialog box with the
//   mouse
//
////////////////////////////////////
//
// include files
//

#include <stdio.h>
#include <stdlib.h>
#include "tproto.h"

////////////////////////////////////
//
// function prototype
//

void FILE_About_ALT_A(void);

////////////////////////////////////
//
// global data

int    mouse_installed= 0;

////////////////////////////////////
//
// about dialog data
//

char    dot[3]=          { '[', 254, ']' };

char    FILE_about_bar[30]= { 199, 196, 196, 196, 196,
                               196, 196, 196, 196, 196,
                               196, 196, 196, 196, 196,
                               196, 196, 196, 196, 196,
                               196, 196, 196, 196, 182 };

char    FILE_about3[28]=   "    Dranoel Software Inc    ";
char    FILE_about4[28]=   "          xxxxxxxxxxxxxx    ";
char    FILE_about5[28]=   "          xxxxxxxxxxxxxxxxxxxxxx    ";

char    FILE_about7[28]=   "    Crafting OS/2 Utilities    ";
char    FILE_about8[28]=   "          And Programmer Tools    ";
char    FILE_about9[29]=   "    For Now and the Future    ";
char    FILE_about32[28]=  "          ";
char    FILE_ok[4]=        " OK ";

void FILE_About_ALT_A()
{
```

```

WIND    *W1;
UCHAR   attr1, attr2;
int      key;

//
// initialize attribute
//

attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);
attr2= mk_attr(BLUE, WHITE, OFF_INTENSITY, OFF_BLINK);

//
// turn off the mouse
//

if(mouse_installed) {
    ms_off();
}

//
// Initialize Window
//

W1 = wind_init(W1,
               4,
               24,
               4+14,
               24+29,
               attr1,
               D_D_D_D,
               "");

//
// write dialog window messages
//

wind_write(W1, 0, 1, 3, dot, W1->attr);
wind_write(W1, 1, 1, 28, FILE_about32, W1->attr);
wind_write(W1, 2, 1, 28, FILE_about3, attr2);
wind_write(W1, 3, 1, 28, FILE_about4, attr2);
wind_write(W1, 4, 1, 28, FILE_about5, attr2);
wind_write(W1, 5, 1, 28, FILE_about32, W1->attr);
wind_write(W1, 6, 0, 30, FILE_about_bar, W1->attr);
wind_write(W1, 7, 1, 28, FILE_about7, attr2);
wind_write(W1, 8, 1, 28, FILE_about8, attr2);
wind_write(W1, 9, 1, 28, FILE_about9, attr2);
wind_write(W1, 10, 0, 30, FILE_about_bar, W1->attr);
wind_write(W1, 11, 1, 28, FILE_about32, W1->attr);
wind_write(W1, 12, 13, 4, FILE_ok, mk_attr_inverse(attr2));

//
// turn on the mouse
//

if(mouse_installed) {
    ms_on();
}

//

```

6-1 Continued.

```
// map the display
//

ms_map_display(20, 0, F10);

//
// turn off the mouse
//

if(mouse_installed) {
    ms_off();
}

//
// remove window and display original screen information
//

wind_remove(W1);

//
// destroy the window structure
//

wind_destroy(W1);

//
// turn on the mouse
//

if(mouse_installed) {
    ms_on();
}

}

//
////////////////////////////////////

////////////////////////////////////
//
// main()
//

void main()
{
    //
    // remove the text cursor
    //

    cu_remove();

    //
    // initialize the screen
    //

    scrn_init();

    //
```



```

// clear the screen
//

scrn_clear();

//
// initialize the mouse
//

mouse_installed= ms_init();

//
// turn on the mouse
//

if(mouse_installed) {
    ms_on();
}

//
// call drop down window function
//

FILE_About_ALT_A();

//
// turn off the mouse
//

if(mouse_installed) {
    ms_off();
}

//
// display the text cursor
//

cu_display();
}

//
////////////////////////////////////

```

6-1 Ends.

An about... dialog box demo program

Figure 6-2 presents the source code listing to PROG6-2.C. This program used the information gained by running PROG6-1.EXE to facilitate input from the keyboard and mouse. Spend time playing with PROG6-1.EXE and PROG6-2.EXE. Spend time looking at the source code presented in PROG6-1.C (FIG. 6-1) and PROG6-2.C (FIG. 6-2). Soon you'll be able to create dialog boxes within a few minutes. This dialog box will be used in PROG7-2.C (FIG. 7-2).

6-2 The source code listing to PROG6-2.C.

```
////////////////////////////////////////
//
// prog6-2.c
//
// Demonstrates:
//   An About... dialog box
//
////////////////////////////////////////
////////////////////////////////////////
//
// include files
//

#include <stdio.h>
#include "tproto.h"

////////////////////////////////////////
//
// global data

int    mouse_installed= 0;

////////////////////////////////////////
//
// about dialog data
//

char    dot[3]=          { '[', 254, ']' };

char    FILE_about_bar[30]= { 199, 196, 196, 196, 196,
                               196, 196, 196, 196, 196,
                               196, 196, 196, 196, 196,
                               196, 196, 196, 196, 196,
                               196, 196, 196, 196, 196,
                               196, 196, 196, 196, 182 };

char    FILE_about3[28]=    "    Dranoel Software Inc    ";
char    FILE_about4[28]=    "    xxxxxxxxxxxxxxxx    ";
char    FILE_about5[28]=    "    xxxxxxxxxxxxxxxxxxxx    ";

char    FILE_about7[28]=    "    Crafting OS/2 Utilities    ";
char    FILE_about8[28]=    "    And Programmer Tools    ";
char    FILE_about9[29]=    "    For Now and the Future    ";
char    FILE_about32[28]=   "    ";
char    FILE_ok[4]=         " OK ";

void FILE_About_ALT_A(void);

void
FILE_About_ALT_A()
{
WIND    *W1;
UCHAR    attr1, attr2, exit_flag= 0;
int      key, x, y;

    //
    // initialize attribute
```

```

//

attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);
attr2= mk_attr(BLUE, WHITE, OFF_INTENSITY, OFF_BLINK);

//
// turn off the mouse
//

if(mouse_installed) {
    ms_off();
}

//
// Initialize Window
//

W1 = wind_init(W1,
               4,
               24,
               4+14,
               24+29,
               attr1,
               D_D_D_D,
               "");

//
// write dialog window messages
//

wind_write(W1, 0, 1, 3, dot, W1->attr);
wind_write(W1, 1, 1, 28, FILE_about32, W1->attr);
wind_write(W1, 2, 1, 28, FILE_about3, attr2);
wind_write(W1, 3, 1, 28, FILE_about4, attr2);
wind_write(W1, 4, 1, 28, FILE_about5, attr2);
wind_write(W1, 5, 1, 28, FILE_about32, W1->attr);
wind_write(W1, 6, 0, 30, FILE_about_bar, W1->attr);
wind_write(W1, 7, 1, 28, FILE_about7, attr2);
wind_write(W1, 8, 1, 28, FILE_about8, attr2);
wind_write(W1, 9, 1, 28, FILE_about9, attr2);
wind_write(W1, 10, 0, 30, FILE_about_bar, W1->attr);
wind_write(W1, 11, 1, 28, FILE_about32, W1->attr);
wind_write(W1, 12, 13, 4, FILE_ok, mk_attr_inverse(attr2));

//
// turn on the mouse
//

if(mouse_installed) {
    ms_on();
}

//
// read the keyboard
//

do {
    // check to see if key press

    key= kb_status();

```

6-2 Continued.

```
// if no key press then read the mouse

if(!key) {

    // check for button press

    key= ms_status(&x, &y);

    // if button press and mouse on the dot

    if((key == 1) && (x == 208) && (y == 32)) {
        key= ENTER;
    }

    // if button press and mouse on OK

    if((key == 1) && (x >= 296) && (x <= 320) && (y == 128)) {
        key= ENTER;
    }

    if(key == ENTER) {
        exit_flag= 1;
    }

} while(!exit_flag);

//
// turn off the mouse
//

if(mouse_installed) {
    ms_off();
}

//
// remove window and display original screen information
//

wind_remove(W1);

//
// destroy the window structure
//

wind_destroy(W1);

//
// turn on the mouse
//

if(mouse_installed) {
    ms_on();
}

}

//
////////////////////////////////////
```

```

////////////////////////////////////
//
// main()
//

void main()
{
    //
    // remove the text cursor
    //

    cu_remove();

    //
    // initialize the screen
    //

    scrn_init();

    mouse_installed= ms_init();

    //
    // turn on the mouse
    //

    if(mouse_installed) {
        ms_on();
    }

    FILE_About_ALT_A();

    //
    // turn off the mouse
    //

    if(mouse_installed) {
        ms_off();
    }

    //
    // display the text cursor
    //

    cu_display();
}

//
////////////////////////////////////

```

6-2 Ends.

An alphanumeric data field entry dialog box demo program

Figure 6-3 presents the source code listing to PROG6-3.C. This program presents the code to a very basic dialog box that gets a string from the keyboard. This dialog box will be used in PROG7-2.C (FIG. 7-2).

6-3 The source code listing to PROG6-3.C.

```
////////////////////////////////////
//
// prog6-3.c
//
// Demonstrates:
//   An alpha-numeric file name
//   entry dialog
//
////////////////////////////////////

////////////////////////////////////
//
// include files
//

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <memory.h>
#include "tproto.h"

char **FILE_Open_ALT_0(int *key, int *entries);

////////////////////////////////////
//
// defines

#define FILE_NAME_LIST_MAX    16

////////////////////////////////////
//
// global data

int    mouse_installed= 0;

////////////////////////////////////
//
// file_open dialog data
//

char    dot[3]=                { '[', 254, ']' };

char    FILE_open3[31]=        " Enter File Name:                ";
char    FILE_entry[31]=        " Is Entry Correct:                [Y] [N] ";
char    FILE_add[31]=          " Add Another File:                [Y] [N] ";
char    FILE_open32[31]=       "                                ";

char **FILE_Open_ALT_0(int *key_val, int *entries)
{
WIND    *W1;
FILE    *fptr;
UCHAR   attr1; //, attr2, attr3;
UCHAR   exit_flag= 0, exit_flag1= 0;
char    *cptr, **cptrp;
int      key;
int      x;
int      y;
```

```

int    file_count= 0;
int    count= 0;

//
// initialize array of pointers
//

cptrptr= (char **)calloc(FILE_NAME_LIST_MAX, sizeof(char *));

//
// initialize the array of pointers to 0
//

for(count= 0; count < FILE_NAME_LIST_MAX; count++) {
    *(cptrptr + (count * sizeof(char *)))= (char *)0;
}

//
// initialize attribute
//

attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);

//
// turn off the mouse
//

if(mouse_installed) {
    ms_off();
}

//
// Initialize Window
//

W1 = wind_init(W1,
               8,
               8,
               4+11,
               72,
               attr1,
               D_D_D_D,
               " Open File For Print ");

//
// write dialog window messages
//

wind_write(W1, 0, 1, 3, dot, W1->attr);
wind_write(W1, 1, 1, 31, FILE_open32, W1->attr);
wind_write(W1, 2, 1, 31, FILE_open3, W1->attr);

//
// turn on the mouse
//

if(mouse_installed) {
    ms_on();
}

```

6-3 Continued.

```
//
// read the keyboard
//

do {

// jump her for new entry

add_entry:

    // allocate memory for file name

    cptr= (char *)malloc(40);

    // place pointer in array of pointers

    *(cptrptr + (file_count * sizeof(char *)))= cptr;

    // initialize file name buffer to 0

    memset(cptr, 0, 40);

// jump here on entry abort

abort_entry:

    // display mouse

    cu_display();

    // if the mouse is installed turn it off

    if(mouse_installed) {
        ms_off();
    }

    // get file name string from keyboard

    key= wind_kb_edit(W1,
                     cptr,
                     2,
                     19,
                     34,
                     UPPER,
                     W1->attr);

    // remove the mouse

    cu_remove();

    // if the key is ENTER process name

    if(key == ENTER) {

        // erase the file name from enter row

        wind_repeat_char(W1, 2, 19, 34, ' ', W1->attr);

        // copy the file name to the live below

        wind_write(W1, 3, 2, 0, "File Entry: ", W1->attr);
```



```

wind_write(W1, 3, 14, 0, cptr, W1->attr);

// test to see if the file can be opened

fptr= fopen(cptr, "r");

// on no print error message and allow user to
// enter another name

if(!fptr) {
    wind_write(W1,
               4,
               2,
               0,
               "Error: File Not Found - ESC to continue",
               mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK));

    // wait for key press

    kb_read();

    // erase error messages

    wind_repeat_char(W1, 3, 2, 50, ' ', W1->attr);
    wind_repeat_char(W1, 4, 2, 50, ' ', W1->attr);
    fclose(fptr);
    goto abort_entry;
}

// close the file

else {
    fclose(fptr);
}

// write the file entry string to the screen

wind_write(W1, 4, 1, 31, FILE_entry, W1->attr);
}

// abort file entry process when wind_kb_edit
// return any key other than ENTER

else {
    goto abort_file_entry_process;
}

// if the mouse is installed then turn it on

if(mouse_installed) {
    ms_on();
}

// initialize loop exit flag

exit_flag1= 0;

// loop for key or mouse response to file name

do {

```

6-3 Continued.

```
// no wait and get key press
key= kb_status();

// if no key press and the mouse is installed
if(!key) && (mouse_installed)) {

    // get the mouse status
    key= ms_status(&x, &y);

    // mouse press on Y
    if((key == 1) && (x == 264) && (y == 96)) {
        key= K_Y;
    }

    // mouse press on N
    if((key == 1) && (x == 296) && (y == 96)) {
        key= K_N;
    }

    // mouse press on close window button
    if((key == 1) && (x == 80) && (y == 64)) {
        key= ESCAPE;
    }

}

// ENTER and Y evoke same action
if(key == ENTER) {
    key= K_Y;
}

// ESCAPE aborts process
if(key == ESCAPE) {
    goto abort_file_entry_process;
}

// allow for either lower case or upper case entry
if((key == K_Y) || (key == K_y)) {
    key= K_Y;
}
if((key == K_N) || (key == K_n)) {
    key= K_N;
}
if((key == K_Y) || (key == K_N)) {
    exit_flag1= 1;
}

// loop until exit flag is set
} while(!exit_flag1);
```

```

// if the mouse is installed turn it off
if(mouse_installed) {
    ms_off();
}

// erase screen messages
wind_write(W1, 2, 19, 31, FILE_open32, W1->attr);
wind_write(W1, 3, 1, 31, FILE_open32, W1->attr);

// abort process on N
if(key == K_N) {
    wind_write(W1, 3, 1, 31, FILE_open32, W1->attr);
    wind_write(W1, 4, 1, 31, FILE_open32, W1->attr);
    goto abort_entry;
}

// file add process
save_or_not:
wind_write(W1, 4, 1, 31, FILE_add, W1->attr);

// if the mouse is installed then turn it on
if(mouse_installed) {
    ms_on();
}

// initialize the exit flag
exit_flag1= 0;
do {
    // no wait and get key press
    key= kb_status();

    // if no key press and the mouse is installed
    if((!key) && (mouse_installed)) {
        // get the mouse status
        key= ms_status(&x, &y);

        // mouse press on Y
        if((key == 1) && (x == 264) && (y == 96)) {
            key= K_Y;
        }

        // mouse press on N
        if((key == 1) && (x == 296) && (y == 96)) {
            key= K_N;
        }
    }
}

```

6-3 Continued.

```
// mouse press on close window button

if((key == 1) && (x == 80) && (y == 64)) {
    key= ESCAPE;
}

}

// ENTER and Y evoke same action

if(key == ENTER) {
    key= K_Y;
}

// ESCAPE aborts process

if(key == ESCAPE) {
    goto abort_file_entry_process;
}

// allow for either lower case or upper case entry

if((key == K_Y) || (key == K_y)) {
    key= K_Y;
}

if((key == K_N) || (key == K_n)) {
    key= K_N;
}

if((key == K_Y) || (key == K_N)) {
    exit_flag1= 1;
}

// loop until exit flag is set

} while(!exit_flag1);

// if the mouse is installed turn it off

if(mouse_installed) {
    ms_off();
}

// on yes process entry and adjust file counter

if( key == K_Y) {
    if(file_count == 14) {
        wind_write(W1,
                    4,
                    2,
                    0,
                    "File Limit Reached - ESC to continue",
                    mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK));
        // wait for key press

        kb_read();

        // erase error messages
```

```

        wind_repeat_char(W1, 3, 2, 50, ' ', W1->attr);
        wind_repeat_char(W1, 4, 2, 50, ' ', W1->attr);

        goto save_or_not;
    }

    wind_write(W1, 3, 1, 31, FILE_open32, W1->attr);
    wind_write(W1, 4, 1, 31, FILE_open32, W1->attr);
    file_count++;
    goto add_entry;
}

// if the mouse is installed turn it on
if(mouse_installed) {
    ms_on();
}

// initialize exit flag
exit_flag= 1;

} while(!exit_flag);

//
// jump here on abort of the file entry process
//

abort_file_entry_process:

*entries= file_count + 1;

//
// return key_val to calling function
//

*key_val= key;

//
// turn off the mouse
//

if(mouse_installed) {
    ms_off();
}

//
// remove window and display original screen information
//

wind_remove(W1);

//
// destroy the window structure
//

wind_destroy(W1);

```

6-3 Continued.

```
//
// turn on the mouse
//

if(mouse_installed) {
    ms_on();
}

return cptrptr;
}

//
////////////////////////////////////

void main()
{
char **cptrptr;
int  ret_val, count;
int  entries;
char *blk;

//
// remove the text cursor
//

cu_remove();

//
// initialize the screen
//

scrn_init();

mouse_installed= ms_init();

//
// turn on the mouse
//

if(mouse_installed) {
    ms_on();
}

cptrptr= FILE_Open_ALT_0(&ret_val, &entries);

//
// turn off the mouse
//

if(mouse_installed) {
    ms_off();
}

//
// display the text cursor
```

```

//
cu_display();

//
// print the cptrptr entered
//

for(count= 0; count < entries; count++) {
    if(*(cptrptr + (count * sizeof(char *))) != 0) {
        printf("\nFile Name: %s", *(cptrptr + (count * sizeof(char *)));
        blk= *(cptrptr + (count * sizeof(char *)));
        free(blk);
    }
}

free((void *)cptrptr);
}

//
////////////////////////////////////

```

6-3 Ends.

A multiple item check dialog box demo program

Figure 6-4 presents the source code to a multiple item check dialog box program. Look at the source code and see how easy it will be to modify this dialog box to suit your own needs. This dialog box will be used in PROG7-2.C (FIG. 7-2).

6-4 The source code listing to PROG6-4.C.

```

////////////////////////////////////
//
// prog6-4.c
//
// Demonstrates:
// Mapping a check box style
// dialog box
//
//
////////////////////////////////////
////////////////////////////////////
//
// include files
//

#include <malloc.h>
#include <stdio.h>

#include "tproto.h"

////////////////////////////////////
//
// global data

```

6-4 Continued.

```
int    mouse_installed= 0;

////////////////////////////////////
//
// about dialog data
//

char    dot[3]=          { '[', 254, ']' };

char    OPTIONS_set_bar[34]= { 199, 196, 196, 196, 196, 196,
                                196, 196, 196, 196, 196, 196,
                                196, 196, 196, 196, 196, 196,
                                196, 196, 196, 196, 196, 196,
                                196, 196, 196, 196, 196,
                                196, 196, 196, 196, 182 };

char    OPTIONS_set2[32]=    "      Options           On Off ";
char    OPTIONS_set4[32]=    " Compress (136 dpi)      [ ] [ ] ";
char    OPTIONS_set5[32]=    " Double Strike       [ ] [ ] ";
char    OPTIONS_set6[32]=    " Header (File & Page) [ ] [ ] ";

char    OPTIONS_set8[32]=    " Left Column Offset  [00] [05] ";
char    OPTIONS_set9[32]=    "   In Blank Spaces   [10] [15] ";
char    OPTIONS_set10[32]=   "                               [20] [25] ";

char    OPTIONS_set12[32]=   "      [ Ok ]           [ Cancel ] ";

typedef struct PrintOpt {
    UCHAR    compress;
    UCHAR    double_strike;
    UCHAR    header;
    UCHAR    left_offset;
};

int OPTIONS_Set_ALT_S(struct PrintOpt *P01)
{
    WIND    *W1;
    UCHAR    attr1, attr2, compress, double_strike, change= 0;
    UCHAR    header, left_offset, exit_flag= 0;
    int      x, y, key;

    //
    // transfer PrintOpt structure values to
    // local variables
    //

    compress= P01->compress;
    double_strike= P01->double_strike;
    header= P01->header;
    left_offset= P01->left_offset;

    //
    // initialize attribute
    //

    attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);
    attr2= mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK);
```



```

//
// turn off the mouse
//

if(mouse_installed) {
    ms_off();
}

//
// Initialize Window
//

W1 = wind_init(W1,
               4,
               20,
               4+13,
               20+33,
               attr1,
               D_D_D_D,
               " Print Options ");

//
// write dialog window messages
//

wind_write(W1, 0, 1, 3, dot, W1->attr);
wind_write(W1, 2, 1, 32, OPTIONS_set2, W1->attr);
wind_repeat_char(W1, 3, 7, 7, 196, W1->attr);
wind_repeat_char(W1, 3, 25, 3, 196, W1->attr);
wind_repeat_char(W1, 3, 29, 3, 196, W1->attr);
wind_write(W1, 4, 1, 32, OPTIONS_set4, W1->attr);
wind_char(W1, 4, 2, 'C', attr2);
wind_write(W1, 5, 1, 32, OPTIONS_set5, W1->attr);
wind_char(W1, 5, 2, 'D', attr2);
wind_write(W1, 6, 1, 32, OPTIONS_set6, W1->attr);
wind_char(W1, 6, 2, 'H', attr2);
wind_write(W1, 7, 1, 32, OPTIONS_set8, W1->attr);
wind_char(W1, 7, 2, 'L', attr2);
wind_write(W1, 8, 1, 32, OPTIONS_set9, W1->attr);
wind_write(W1, 9, 1, 32, OPTIONS_set10, W1->attr);
wind_write(W1, 11, 1, 32, OPTIONS_set12, W1->attr);

//
// display current setup
//

wind_char(W1, 11, 8, '0', attr2);
wind_char(W1, 11, 22, 'C', attr2);

if(compress) {
    wind_char(W1, 4, 26, 'X', attr2);
}

else {
    wind_char(W1, 4, 30, 'X', attr2);
}

if(double_strike) {

```

6-4 Continued.

```
        wind_char(W1, 5, 26, 'X', attr2);
    }

    else {
        wind_char(W1, 5, 30, 'X', attr2);
    }

    if(header) {
        wind_char(W1, 6, 26, 'X', attr2);
    }

    else {
        wind_char(W1, 6, 30, 'X', attr2);
    }

    switch(left_offset) {
        case 00:
            wind_attr(W1, 7, 24, 2, attr2);
            break;
        case 10:
            wind_attr(W1, 8, 24, 2, attr2);
            break;
        case 20:
            wind_attr(W1, 9, 24, 2, attr2);
            break;
        case 05:
            wind_attr(W1, 7, 29, 2, attr2);
            break;
        case 15:
            wind_attr(W1, 8, 29, 2, attr2);
            break;
        case 25:
            wind_attr(W1, 9, 29, 2, attr2);
            break;

    }

    //
    // turn on the mouse
    //

    if(mouse_installed) {
        ms_on();
    }

    //
    // process keyboard and mouse input
    //

    exit_flag= 0;

    do {
        if(change) {
            change= 0;

            if(mouse_installed) {
                ms_off();
            }
        }
    }
```

```

    if(compress) {
        wind_char(W1, 4, 26, 'X', attr2);
        wind_char(W1, 4, 30, ' ', W1->attr);
    }
    else {
        wind_char(W1, 4, 26, ' ', W1->attr);
        wind_char(W1, 4, 30, 'X', attr2);
    }

    if(double_strike) {
        wind_char(W1, 5, 26, 'X', attr2);
        wind_char(W1, 5, 30, ' ', W1->attr);
    }
    else {
        wind_char(W1, 5, 26, ' ', W1->attr);
        wind_char(W1, 5, 30, 'X', attr2);
    }

    if(header) {
        wind_char(W1, 6, 26, 'X', attr2);
        wind_char(W1, 6, 30, ' ', W1->attr);
    }
    else {
        wind_char(W1, 6, 26, ' ', W1->attr);
        wind_char(W1, 6, 30, 'X', attr2);
    }

    wind_attr(W1, 7, 24, 2, W1->attr);
    wind_attr(W1, 8, 24, 2, W1->attr);
    wind_attr(W1, 9, 24, 2, W1->attr);
    wind_attr(W1, 7, 29, 2, W1->attr);
    wind_attr(W1, 8, 29, 2, W1->attr);
    wind_attr(W1, 9, 29, 2, W1->attr);

    switch(left_offset) {
        case 00:
            wind_attr(W1, 7, 24, 2, attr2);
            break;
        case 10:
            wind_attr(W1, 8, 24, 2, attr2);
            break;
        case 20:
            wind_attr(W1, 9, 24, 2, attr2);
            break;
        case 05:
            wind_attr(W1, 7, 29, 2, attr2);
            break;
        case 15:
            wind_attr(W1, 8, 29, 2, attr2);
            break;
        case 25:
            wind_attr(W1, 9, 29, 2, attr2);
            break;
    }
    if(mouse_installed) {
        ms_on();
    }
}

key= kb_status();

```

6-4 Continued.

```
if((!key) && (mouse_installed)) {
    key= ms_status(&x, &y);

    if((key == 1) && (x >= 208) && (x <= 248) && (y == 120)) {
        key= ENTER;
    }

    if((key == 1) && (x >= 320) && (x <= 392) && (y == 120)) ||
        ((x == 176) && (y == 32))) {
        key= ESCAPE;
    }

    // Compress off to on

    if((key == 1) && (x == 368) && (y == 64)) {
        if(!compress) {
            key= ALT_C;
        }
    }

    // Compress on to off

    if((key == 1) && (x == 400) && (y == 64)) {
        if(compress) {
            key= ALT_C;
        }
    }

    // Double strike off to on

    if((key == 1) && (x == 368) && (y == 72)) {
        if(!double_strike) {
            key= ALT_D;
        }
    }

    // Double strike on to off

    if((key == 1) && (x == 400) && (y == 72)) {
        if(double_strike) {
            key= ALT_D;
        }
    }

    // header off to on

    if((key == 1) && (x == 368) && (y == 80)) {
        if(!header) {
            key= ALT_H;
        }
    }

    // header on to off

    if((key == 1) && (x == 400) && (y == 80)) {
        if(header) {
            key= ALT_H;
        }
    }
}
```

```

        if((key == 1) && (x >= 352) && (x <= 360) && (y == 88)) {
            left_offset= 00;
            change= 1;
        }
        if((key == 1) && (x >= 392) && (x <= 400) && (y == 88)) {
            left_offset= 05;
            change= 1;
        }
        if((key == 1) && (x >= 352) && (x <= 360) && (y == 96)) {
            left_offset= 10;
            change= 1;
        }
        if((key == 1) && (x >= 392) && (x <= 400) && (y == 96)) {
            left_offset= 15;
            change= 1;
        }
        if((key == 1) && (x >= 352) && (x <= 360) && (y == 104)) {
            left_offset= 20;
            change= 1;
        }
        if((key == 1) && (x >= 392) && (x <= 400) && (y == 104)) {
            left_offset= 25;
            change= 1;
        }
    }

    if(key == ALT_C) {
        if(!compress) {
            compress= 1;
        }

        else {
            compress= 0;
        }
        change= 1;
    }

    if(key == ALT_D) {
        if(!double_strike) {
            double_strike= 1;
        }

        else {
            double_strike= 0;
        }
        change= 1;
    }

    if(key == ALT_L) {
        if(left_offset == 25) {
            left_offset= 0;
        }

        else {
            left_offset+= 5;
        }
        change= 1;
    }

    if(key == ALT_H) {
        if(!header) {

```

6-4 Continued.

```
        header= 1;
    }

    else {
        header= 0;
    }
    change= 1;
}

if((key == ESCAPE) || (key == ENTER)){
    exit_flag= 1;
}

} while(!exit_flag);

//
// if key is equal to ENTER then
// transfer the data from the
// local shadow of the
// PrintOpt structure to the
// parameter PrintOpt pointer's
// structure
//

if(key == ENTER) {
    P01->compress= compress;
    P01->double_strike= double_strike;
    P01->header= header;
    P01->left_offset= left_offset;
}

//
// turn off the mouse
//

if(mouse_installed) {
    ms_off();
}

//
// remove window and display original screen information
//

wind_remove(W1);

//
// destroy the window structure
//

wind_destroy(W1);

//
// turn on the mouse
//
```

```

        if(mouse_installed) {
            ms_on();
        }

        return key;
    }

    //
    //////////////////////////////////////
    //////////////////////////////////////
    //
    // main()
    //

void main()
{
    struct PrintOpt P01;
    int    key;

    //
    // initialize print options members
    //

    P01.compress=    0;
    P01.double_strike= 0;
    P01.header=      0;
    P01.left_offset=  0;

    //
    //
    // remove the text cursor
    //

    cu_save_loc();
    cu_remove();

    //
    // initialize the screen
    //

    scrn_init();

    //
    // initialize the mouse
    //

    mouse_installed= ms_init();

    //
    // turn on the mouse
    //

    if(mouse_installed) {
        ms_on();
    }

    key= OPTIONS_Set_ALT_S(&P01);

```

6-4 Continued.

```
//
// turn off the mouse
//

if(mouse_installed) {
    ms_off();
}

//
// display the text cursor
//

cu_display();

//
// report the status of the
// Print Option dialog box
//

cu_rest_loc();

if(key == ESCAPE) {
    printf("\nESCAPE returned\n");
}

else {
    printf("\nENTER returned\n");
}

printf("P01.compress= %d\n", P01.compress);
printf("P01.double_strike= %d\n", P01.double_strike);
printf("P01.header= %d\n", P01.header);
printf("P01.left_offset= %d\n", P01.left_offset);

}

//
////////////////////////////////////
```

6-4 Ends.

A string list dialog box

Figure 6-5 presents the source code listing to PROG6-5.C. This program shows how to list a variety of file names to a dialog box. This dialog box will be used in PROG7-2.C (FIG. 7-2).

6-5 The source code listing to PROG6-5.C.

```
////////////////////////////////////
//
// prog6-5.c
//
// Demonstrates:
//
// Display file names from char **
// list
//
////////////////////////////////////
```



```

////////////////////////////////////
//
// include files
//

#include <stdio.h>
#include "tproto.h"

////////////////////////////////////
//
// global data

int    mouse_installed= 0;

////////////////////////////////////
//
// display file names
//

WIND *FILE_display_files(char **files, int entries)
{
WIND    *W1;
UCHAR   attr1;
int     key, counter;

    //
    // initialize attribute
    //

    attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);

    //
    // Initialize Window
    //

    W1 = wind_init(W1,
                    2,
                    40,
                    22,
                    79,
                    attr1,
                    D_D_D_D,
                    " File Print List ");

    //
    // write dialog window messages
    //

    wind_write(W1, 2, 2, 9, "File Name", W1->attr);
    wind_repeat_char(W1, 3, 2, 9, 196, W1->attr);
    wind_write(W1, 2, 26, 12, "Print Status", W1->attr);
    wind_repeat_char(W1, 3, 26, 12, 196, W1->attr);

    for(counter= 0; counter < entries; counter++) {
        wind_write(W1, 4 + counter, 2, 0, (char *)files[counter], W1->attr);
        wind_write(W1, 4 + counter, 26, 0, "Waiting...", W1->attr);
    }
}

```

6-5 Continued.

```
    return(W1);
}

//
////////////////////////////////////
////////////////////////////////////
//
// main()
//

void main()
{
WIND *Wptr;
char *file[15] = { "PROG6-1.C",
                  "PROG6-2.C",
                  "PROG6-3.C",
                  "PROG6-4.C",
                  "PROG6-5.C",
                  "PROG2-1.C",
                  "PROG2-2.C",
                  "PROG2-3.C",
                  "PROG7-1.C",
                  "PROG7-2.C",
                  "..\\..\\FIG2-3.DOC",
                  "MAKEFILE",
                  "MENU.MAK",
                  "TPROTO.H",
                  "TSTRUCT.H" } ;

//
// remove the text cursor
//

cu_remove();

//
// initialize the screen
//

scrn_init();

mouse_installed= ms_init();

Wptr= FILE_display_files(file, 15);

//
// wait for key press
//

kb_read();

//
// remove the window
//

wind_remove(Wptr);
wind_destroy(Wptr);
```

```

//
// turn off the mouse
//

if(mouse_installed) {
    ms_off();
}

//
// display the text cursor
//

cu_display();
}

//
////////////////////////////////////

```

6-5 Ends.

A file name and file size dialog box

Figure 6-6 presents the source code listing to PROG6-6.C. This program displays a dialog box that presents a filename and shows the file size counting from 1 to the file length. This dialog box will be used in PROG7-2.C (FIG. 7-2).

6-6 The source code listing to PROG6-6.C.

```

////////////////////////////////////
//
// prog6-6.c
//
// Demonstrates:
//   Print file name window
//
////////////////////////////////////

////////////////////////////////////
//
// include files
//

#include <stdio.h>
#include <stdlib.h>
#include "tproto.h"

////////////////////////////////////
//
// global data

int    mouse_installed= 0;

////////////////////////////////////
//
// print display of file being
// printed
//

```

6-6 Continued.

```
WIND *PRINT_Start_ALT_S(char *file_name)
{
WIND    *W1;
UCHAR   attr1, attr2, exit_flag;
int      key, x, y;

    //
    // initialize attribute
    //

    attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);
    attr2= mk_attr(BLUE, WHITE, OFF_INTENSITY, OFF_BLINK);

    //
    // Initialize Window
    //

    W1 = wind_init(W1,
                    4,
                    0,
                    4+7,
                    39,
                    attr1,
                    D_D_D_D,
                    " Print Files ");

    //
    // write dialog window messages
    //

    wind_write(W1, 2, 1, 0, " File Name: ", attr2);
    wind_write(W1, 2, 12, 0, file_name, attr2);
    wind_write(W1, 3, 1, 0, " File Size: ", attr2);
    wind_write(W1, 4, 1, 0, " Number of Chars Printed: ", attr2);

    return(W1);
}

//
////////////////////////////////////
////////////////////////////////////
//
// main()
//

void main()
{
WIND    *Wptr;
char    buffer[20];
int      counter;
UCHAR   attr1, attr2;

    //
    // make attribute
    //
```

```

    attr1= mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK);
    attr2= mk_attr(BLUE, WHITE, OFF_INTENSITY, OFF_BLINK);

    //
    // remove the text cursor
    //

    cu_remove();

    //
    // initialize the screen
    //

    scrn_init();

    Wptr= PRINT_Start_ALT_S("PROG6-6.C");

    wind_write(Wptr, 3, 13, 0, (char *)itoa(25000, buffer, 10), attr1);

    for(counter= 0; counter < 25000; counter++) {
        wind_write(Wptr, 4, 27, 0, (char *)itoa(counter + 1, buffer, 10),
attr1);
    }

    wind_write(Wptr, 5, 1, 0, " File Print Job Complete ", attr2);

    kb_read();

    wind_remove(Wptr);
    wind_destroy(Wptr);

    //
    // display the text cursor
    //

    cu_display();
}

//
//////////

```

6-6 Ends.

A Lotus grid style menu demonstration program

Figure 6-7 presents the source code listing to PROG6-7.C. This program demonstrates how to create a scroll bar menu along with a grid style menu. Code from this program may be used to fashion menus in your own programs. All you need to do is change the number of menu items and the names of the menu items.

6-7 The source code listing to PROG6-7.C.

```

/*****
/* prog6-7.c          */
/*                  */
/* Menu Demonstration*/
/*                  */
/* OS/2 & DOS C      */
/* Toolkit           */

```

[illegible]

```

196,196,196,196,196,196,196,196,196,
196,196,196,180 };
char item1[29] = " Lotus Style Menu      ";
char item2[29] = " Grid Style Menu      ";
char item3[29] = " Some Historical Information ";
char item5[29] = " Quit OS/2 & DOS Demo      ";

/* Messages for LOTUS Window */

char menu1[47] = " Mean Mode Median Range Standard Deviation ";
char mess1[47] = " Mean is the Average score of the distribution ";
char mess2[47] = " Mode is the most frequent score      ";
char mess3[47] = " Median is the middle score of sample   ";
char mess4[47] = " Range is the distance from highest to lowest ";
char mess5[47] = " Standard dev. is avg. distance from mean ";

/* lot_map holds mess column offset & length */

int lot_map[5][2] = {
    1,6,
    7,6,
    13,8,
    21,7,
    28,20 };

/* messages for GRID window - holds row & column */

char gmenu[21] = " SELECT A NUMBER ";
char grid1[21] = " 1 2 3 ";
char grid2[21] = " 4 5 6 ";
char grid3[21] = " 7 8 9 ";
char grid4[21] = " Press ENTER to Exit ";

/* grid_map row,column for start of inverse item */

int grid_map[9][2] = {
    3,7,
    3,10,
    3,13,
    4,7,
    4,10,
    4,13,
    5,7,
    5,10,
    5,13 };

/* info1 window data */

char speed1[28] = " OS/2 & DOS C Library ";
unsigned char speed2[30] = { 199,196,196,196,196,196,196,196,
    196,196,196,196,196,196,196,196,
    196,196,196,196,196,196,196,196,
    196,196,182 };
char speed3[28] = " Dranoel Software Inc ";
char speed4[28] = " ----- ";
char speed5[28] = " xxxxxxxx ";
char speed6[28] = " xxxxxxxxxxxxxxxxxxxxxxxx ";
char speed7[28] = " ----- ";
char speed8[28] = " Press ANY KEY to exit. ";

```

6-7 Continued.

```

/*****/
/* global variables*/
/*****/

int xinverse;          /* attribute for inverse      */
int hl_tense;          /* highlight bar intensity    */

/*****/
/*
/* Lotus Style Window      */
/*
/* Receives: nothing      */
/* Returns: item selection number */
/*
/* Displays Lotus style window */
/* with attendant cursor, high- */
/* light and item description */
/* routines.                */
/*
/*
/*****/

int
tlotus()
{
int key;  /* scan and char value */
int exit; /* val for loop cond chk */
int exp_a; /* item explanation attr */

/*****/
/* Initialize lotus menu window structure and display window */
/*****/

/* Set lotus explanation Attr - Fore,Back,Intensity,Blink */
exp_a = mk_attr(MAGENTA,BLUE,ON_INTENSITY,OFF_BLINK);

/* call window initialization routines only once */

if(!lotus_flag)
{
    /* ensure window startup bypassed next window call */
    lotus_flag=1;

    /* Allocate memory and return pointer to structure */
    LOTUS = setWind(LOTUS,6,20,9,68);

    /* Set Window Attr - Fore,Back,Intensity,Blink */
    setAttr(LOTUS,mk_attr(WHITE,BLUE,ON_INTENSITY,OFF_BLINK));

    /* Set Window Border - top, bot, left, right */
    setBord(LOTUS,S_S_S_S);

    /* Set the top and bottom title - 0 set no bottom title */
    setTitle(LOTUS," Lotus Style Window ");
}
```



```

        /* Display window */
        strtWind(LOTUS);
    }
else
    wind_display(LOTUS);

/* set loop condition */
exit=aFALSE;

do
{
    /* Write title bar - erasing old inverse */
    wind_write(LOTUS,1,1,47,menu1,LOTUS->attr);

    /* Inverse proper menu item using lot_map[][] */
    wind_attr(LOTUS,1,lot_map[lotus_item][0],
              lot_map[lotus_item][1],
              hl_tense);

    switch(lotus_item)      /* print item explanation */
    {
        case 0:
            wind_write(LOTUS,2,1,47,mess1,exp_a);
            break;
        case 1:
            wind_write(LOTUS,2,1,47,mess2,exp_a);
            break;
        case 2:
            wind_write(LOTUS,2,1,47,mess3,exp_a);
            break;
        case 3:
            wind_write(LOTUS,2,1,47,mess4,exp_a);
            break;
        case 4:
            wind_write(LOTUS,2,1,47,mess5,exp_a);
            break;
    }

    key = kb_read();

    switch(key)
    {
        case RIGHT_ARROW:      /* At right item? */
        case RIGHT_ARROW_K:    /* At right item? */
            if(lotus_item==4)  /* Yes? */
                lotus_item=0;  /* set left item */
            else                /* Else */
                lotus_item++;   /* move rt 1 item */
            break;

        case LEFT_ARROW:       /* At left item? */
        case LEFT_ARROW_K:     /* At left item? */
            if(lotus_item==0)  /* Yes? */
                lotus_item=4;  /* set right item */
            else                /* Else */
                lotus_item--;   /* move lft 1 item */
            break;
    }
}

```

6-7 Continued.

```
        case ENTER:
            exit=aTRUE;
            break;
    }

    } while(!exit);

    /* Remove Lotus Window */

    wind_remove(LOTUS);

    /* return selected item number */

    return(lotus_item);
}

/*****
/*
/* Grid Style Window
/*
/* Receives: nothing
/* Returns: item selection number
/*
/* Displays Grid style window
/* with attendant cursor & high-
/* light description routines.
/*
*****/

/*****
/* Make variables that must retain their
/* value after the function exits global
*****/

int
tgrid()
{
    int key; /* scan and char value */
    int exit; /* val for loop cond chk */

    /*****
    /* Initialize grid menu window structure and display window
    *****/

    if(!grid_flag)
    {
        /* ensure window initialization bypass */

        grid_flag=1;

        /* Allocate memory and return pointer to structure */

        GRID = setWind(GRID,10,10,18,32);

        /* Set Window Attr - Fore,Back,Intensity,Blink */

        setAttr(GRID,mk_attr(WHITE,RED,OFF_INTENSITY,OFF_BLINK));

        /* Set Window Border */
```

```

        setBord(GRID,D_D_D_D);

        /* Set the top and bottom title - 0 set no bottom title */
        setTitle(GRID," Grid Style Window ");

        /* Display window */

        strtWind(GRID);
    }

else
    wind_display(GRID);

/* Write name and exit messages */

wind_write(GRID,1,1,21,gmenu,xinverse);
wind_write(GRID,7,1,21,grid4,GRID->attr);

wind_write(GRID,7,8,5,"ENTER",mk_attr(WHITE,RED,OFF_INTENSITY,ON_BLINK));

/* set loop condition */

exit=aFALSE;

do
{
    /* Write grid entries bar */

    wind_write(GRID,3,1,21,grid1,GRID->attr);
    wind_write(GRID,4,1,21,grid2,GRID->attr);
    wind_write(GRID,5,1,21,grid3,GRID->attr);

    /* Inverse proper menu item using grid_map[][] */

    wind_attr(GRID,grid_map[grid_item][0],
               grid_map[grid_item][1],
               3,
               xinverse);

    key = kb_read();

    switch(key)
    {
        case RIGHT_ARROW:
        case RIGHT_ARROW_K:
            /* IF rt col->mv to left col ELSE->mv rt */

            if( (grid_item==0) || (grid_item==1) ||
                (grid_item==3) || (grid_item==4) ||
                (grid_item==6) || (grid_item==7) )
                grid_item++;

            else if(grid_item==2)
                grid_item=0;

            else if(grid_item==5)
                grid_item=3;

            else
                grid_item=6;
            break;
    }
}

```

6-7 Continued.

```
case LEFT_ARROW:
case LEFT_ARROW_K:
/* IF left col->mv to rt col ELSE->mv left */

    if( (grid_item==2)|| (grid_item==1)||
        (grid_item==5)|| (grid_item==4)||
        (grid_item==8)|| (grid_item==7) )
        grid_item--;

    else if(grid_item==0)
        grid_item=2;

    else if(grid_item==3)
        grid_item=5;

    else
        grid_item=8;
    break;

case DOWN_ARROW:
case DOWN_ARROW_K:
/* IF bottom row->mv to top row ELSE->mv down */

    if(grid_item<=5)
        grid_item += 3;

    else if(grid_item==6)
        grid_item=0;

    else if(grid_item==7)
        grid_item=1;

    else
        grid_item=2;
    break;

case UP_ARROW:
case UP_ARROW_K:
/* IF top row->mv to bottom row ELSE->mv up */

    if(grid_item>=3)
        grid_item -= 3;

    else if(grid_item==0)
        grid_item=6;

    else if(grid_item==1)
        grid_item=7;

    else
        grid_item=8;
    break;
case ENTER:
    exit=aTRUE;
    break;
}
} while(!exit);

/* Remove Lotus Window */

wind_remove(GRID);
```

```

        /* return selected item */
        return(grid_item);
    }

    /***/
    /*
    /* Simple Style Window
    /*
    /* Receives: nothing
    /* Returns: nothing
    /*
    /* Displays Simple pop up
    /* information window.
    /*
    /*
    /***/

    /***/
    /* Make variables that must retain their
    /* value after the function exits global
    /***/

    int infol_flag=0;

    void
    infol()
    {
        /***/
        /* Initialize grid menu window structure and display window */
        /***/

        if(!infol_flag)
        {
            /* ensure window initialization bypass */

            infol_flag=1;

            /* Allocate memory and return pointer to structure */

            INFORM = setWind(INFORM,12-5,20-5,22-5,49-5);

            /* Set Window Attr - Fore,Back,Intensity,Blink */

            setAttr(INFORM,mk_attr(BLACK,CYAN,OFF_INTENSITY,OFF_BLINK));

            /* Set Window Border */

            setBord(INFORM,D_D_D_D);

            /* Set the bottom title */

            setTitle(INFORM," Program History ");

            /* Display window */

            strtWind(INFORM);
        }

        else
            wind_display(INFORM);
    }

```

6-7 Continued.

```
/* Write menu and exit messages */

wind_write(INFORM,1,1,28,speed1,mk_attr(CYAN,
                                         BLACK,
                                         OFF_INTENSITY,
                                         OFF_BLINK));

wind_write(INFORM,2,0,30,speed2,INFORM->attr);
wind_write(INFORM,3,1,28,speed3,INFORM->attr);
wind_write(INFORM,4,1,28,speed4,INFORM->attr);
wind_write(INFORM,5,1,28,speed5,INFORM->attr);
wind_write(INFORM,6,1,28,speed6,INFORM->attr);
wind_write(INFORM,7,1,28,speed7,INFORM->attr);
wind_write(INFORM,8,0,30,speed2,INFORM->attr);
wind_write(INFORM,9,1,28,speed8,INFORM->attr);

/* wait for key press */

kb_read();

/* remove window and display original screen information */

wind_remove(INFORM);
}

/*****
/*
/* int main(void)
/*
/*
/* Receives: nothing
/* Returns: nothing
/*
/* Sets up the FIRST window
/* display and contains the
/* scroll bar menu selection
/* routine.
/*
/*
*****/

int
main()
{
int key;      /* recieves Scan & char key code */
int exit;     /* holds val for main loop check */
int old_row;  /* Tracker for highlight bar
int row;      /* Tracker for highlight bar
int intense;  /* intensity attribute value
int beep;     /* flag for beep on 'Q' keypress

/*****
/* Initialize VIDIO structure*/
/*
/* ALWAYS call at prog start!*/
*****/

scrn_init();

/* Set global attribute intense for inverse video */

xinverse = mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK);
```

```

/* set global attribute hl_tense for WHITE,WHITE,INTENSE,OFF_BLINK */
hl_tense = mk_attr(WHITE,WHITE,ON_INTENSITY,OFF_BLINK);

/* Set intense text attribute for this window */
intense = mk_attr(WHITE,MAGENTA,ON_INTENSITY,OFF_BLINK);

/* Turn off the cursor */
cu_remove();

/*****
/* Initialize main menu window structure and display window */
*****/

/* Allocate memory and return pointer to structure */
FIRST = setWind(FIRST,2,4,10,34);

/* Set Window Attr - Fore,Back,Intensity,Blink */
setAttr(FIRST,mk_attr(WHITE,MAGENTA,OFF_INTENSITY,OFF_BLINK));

/* Set Window Border - top, bot, left, right */
setBord(FIRST,D_D_S_S);

/* Set the top and bottom title */
setTitle(FIRST," Dranoel Software ");

/* Display window */
strtWind(FIRST);

/* Write menu name & line below to window */
wind_write(FIRST,1,1,29,title,xinverse);
wind_write(FIRST,2,0,31,i_bar,FIRST->attr);

/* Write menu items to window */
wind_write(FIRST,3,1,29,item1,FIRST->attr);
wind_write(FIRST,4,1,29,item2,FIRST->attr);
wind_write(FIRST,5,1,29,item3,FIRST->attr);
wind_write(FIRST,6,0,31,i_bar,FIRST->attr);
wind_write(FIRST,7,1,29,item5,FIRST->attr);

/* highlight first letter of item */
wind_attr(FIRST,3,2,1,intense); /* L intense */
wind_attr(FIRST,4,2,1,intense); /* G intense */
wind_attr(FIRST,5,2,1,intense); /* S intense */
wind_attr(FIRST,7,2,1,intense); /* Q intense */

/* Set highlight trackers to start at item1 (row 3) */
row = 3;
old_row = 3;

```

6-7 Continued.

```
/* set default for no beep */
beep = aFALSE;
/* Set loop condition */

exit = aFALSE;

/*****
/* Main keyboard loop. Selects: tlotus(), tgrid(), */
/*                               infol(), & quits */
/* Up,Down arrow or First letter move highlight bar*/
*****/

do
{
    wind_attr(FIRST,old_row,1,29,FIRST->attr); /* off highlight bar */
    wind_attr(FIRST,old_row,2,1,intense);      /* intense item let */
    wind_attr(FIRST,row,1,29,xinverse);        /* on highlight bar */
    wind_attr(FIRST,row,2,1,h1_tense);         /* intense HB letter */
    if(beep)                                   /* YES? beep after */
    {                                           /* scrn update */
        bleep();                             /* Yes-warning beep */
        beep=aFALSE;                         /* reset-> no beep */
    }
    old_row = row;                            /* reset OFF tracker */
    key = kb_read();                          /* get scan & char */
    switch(key)                                /* eval key press */
    {
        case DOWN_ARROW:
        case DOWN_ARROW_K:
            if(row==7)                        /* If bottom row */
                row=3;                        /* then->top row */
            else if(row==5)                   /* If row 5 */
                row=7;                        /* then skip to 7 */
            else                               /* Otherwise */
                row++;                         /* then down row */
            break;
        case UP_ARROW:
        case UP_ARROW_K:
            if(row==7)                        /* If bottom row */
                row=5;                        /* then skip to 5 */
            else if(row==3)                   /* If row 3 */
                row=7;                        /* then->bot row */
            else                               /* Otherwise */
                row--;                         /* then up row */
            break;
        case ENTER:
            switch(row)                       /* Eval selection */
            {
                case 3:                       /* sel. lotus demo */
                    tlotus();
                    break;
                case 4:                       /* sel. grid demo */
                    tgrid();
                    break;
                case 5:                       /* simple demo */
                    infol();
                    break;
                case 7:                       /* Exit option */
                    exit=aTRUE;
            }
    }
}
```



```

        break;
    }
    break;
default:
    key &= 0x00ff; /* Check ascii val */
    switch(key)    /* mask scan code */
    {              /* which key? */
        case 'l': /* L->lotus choice */
        case 'L':
            row=3;
            break;
        case 'g': /* G->grid choice */
        case 'G':
            row=4;
            break;
        case 's': /* S->simple demo */
        case 'S':
            row=5;
            break;
        case 'q': /* Q->quit wind */
        case 'Q':
            row=7;
            beep=aTRUE; /* set for beep */
            break;
    }
    break;
}
} while (!exit);

/* remove window and restore original screen */
wind_remove(FIRST);

/* turn on the cursor */

cu_display();
return(0);
}

```

6-7 Ends.

A commercial quality setup program

Figures 6-8 and 6-9 present the setup program I wrote a few years ago for a ram resident viewer of Dbase and Paradox data files. Although the ram resident database file viewer never made it to market for a variety of complex reasons, the setup program is still pretty neat (in my humble opinion, of course).

The setup program consists of a mini-display of an early version of Borland's Paradox along with a means to control all the colors in the display. The idea was that the ram resident database viewer could be visually configured to the user's whim.

The source for this substantial program is presented in two files. Figure 6-8 presents the source code listing to PROG6-8.C and FIG. 6-9 presents the source code listing to DFINST1.C.

6-8 The source code listing to PROG6-8.C.

```

/*****
/*
/* prog6-8.c
/*
/* DFINST.C
/*
/*
/*
/*
*****/

/*****/
/* Include Files*/
/*****/

#ifdef OS2_PROG

#define INCL_DOSFILEMGR
#include <os2.h>

#endif

#include <io.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <stdio.h>
#include <memory.h>

/*****/
/* C function prototypes*/
/* C-erious Lib Routines*/
/*****/

#include "tproto.h"

/*****/
/* C function prototypes */
/* Routines used by this demo */
/*****/

int tgrid1(void); /* display grid type window */
int tgrid2(void); /* display grid type window */
int tgrid3(void); /* display grid type window */
int tgrid4(void); /* display grid type window */
int tgrid5(void); /* display grid type window */
int tgrid6(void); /* display grid type window */
int tgrid7(void); /* display grid type window */
int tgrid8(void); /* display grid type window */
int tgrid9(void); /* display grid type window */
int tgrid10(void); /* display grid type window */
int tgrid11(void); /* display grid type window */
int tgrid12(void); /* display grid type window */
int tgrid13(void); /* display grid type window */
int tgrid14(void); /* display grid type window */
int info1(void); /* simple pop-up information window */
int tlotus(void); /* display lotus style window */
void main(void); /* program main */
void write_screen(void);
void bleep(void);
void draw_df(void);
```

```

void draw_window_border(void);
void draw_table_top(void);
void remove_df(void);
void draw_mouse_bar(void);
void draw_column_name(void);
void draw_all_data(void);
void draw_resize(void);
void draw_box(int);
void erase_box(int);
void update_tta(int);
void update_tta_back(int);
void update_mba(int);
void update_mba_back(int);
void update_na(int);
void update_na_back(int);
void update_wb(int);
void update_wb_back(int);
void update_ra(int);
void update_ra_back(int);
void update_bwb(int);
void update_bwb_back(int);
void update_ttia(int);
void update_ttia_back(int);
void data_to_iobuff(void);
void iobuff_to_data(void);
void write_cfg(void);
void display_colors(void);
void draw_window_back(void);
void draw_back_wind_bord(void);
void io_to_dup(void);
void dup_to_io(void);

/*****
/* Structure Declarations*/
*****/

struct DF500 {
    char    main_color_attrib;
    char    main_inverse_attrib;
    char    background_window_attrib;
    char    negative_value_attrib;
    char    resize_box_attrib;
    char    mouse_bar_attrib;
    char    passive_window_border_attrib;
    char    mode_flag;      /* 0=25 line, 1=43 line */
    char    catalog_flag; /* 0=auto open off, 1=auto open on */
} header;

/*****
/* Make variables that must retain their */
/* value after the function exits, global */
*****/

/* DF500.CFG buffer
 * iobuff[0] starts DF500 configure data
 * iobuff[64] starts DFINST setup data
 */

char iobuff[256];
char dupbuff[256];
int handle;

```

6-8 Continued.

```
int lotus_flag=0;
int lotus_item=0;
int A1,A2,A3,A4,A5,A6,A7,A8;
int A9,A10,A11,A12,A13,A14,A15,A16;
int col_chart[8] = { BLACK,BLUE,GREEN,CYAN,RED,MAGENTA,BROWN,WHITE };
int grid_flag=0;

/*****
/* Global attribute variables */
*****/

int mode_toggle=0; /* 0=25 line 1=43 line */
int catalog_toggle=0; /* 0=auto open off 1=on */
int table_top_attr;
int tta_fore;
int tta_back;
int tta_flag;
int table_top_inv_attr;
int ttia_fore;
int ttia_back;
int ttia_flag;
int window_background;
int wb_fore;
int wb_back;
int wb_flag;
int resize_attr;
int ra_fore;
int ra_back;
int ra_flag;
int negative_attr;
int na_fore;
int na_back;
int na_flag;
int data_attr;
int da_fore;
int da_back;
int da_flag;
int mouse_bar_attr;
int mba_fore;
int mba_back;
int mba_flag;
int back_wind_bord;
int bwb_fore;
int bwb_back;
int bwb_flag;

int grid_item=0; /* item selected value */
int grid_item1=0; /* item selected value */
int grid_item2=0; /* item selected value */
int grid_item3=0; /* item selected value */
int grid_item4=0; /* item selected value */
int grid_item5=0; /* item selected value */
int grid_item6=0; /* item selected value */
int grid_item7=0; /* item selected value */
int grid_item8=0; /* item selected value */
int grid_item9=0; /* item selected value */
int grid_item10=0; /* item selected value */
int grid_item11=0; /* item selected value */
int grid_item12=0; /* item selected value */
```

```

int grid_item13=0; /* item selected value */
int grid_item14=0; /* item selected value */
int old_grid_item;
int old_grid_item1;
int old_grid_item2;
int old_grid_item3;
int old_grid_item4;
int old_grid_item5;
int old_grid_item6;
int old_grid_item7;
int old_grid_item8;
int old_grid_item9;
int old_grid_item10;
int old_grid_item11;
int old_grid_item12;
int old_grid_item13;
int old_grid_item14;

/*****
/* Global df display data */
*****/

char data_border1[44] = {
    201,91,254,93,205,
    205,205,205,205,205,205,205,205,205,205,
    205,205,205,205,205,205,205,205,205,205,
    205,205,205,205,205,205,205,205,205,205,
    91,24,93,205,91,25,93,205,187 };

char table_top1[] = "Table:
char table_top2[] = "
char table_top3[] = "Masterbk";

/* Pointers to Window Structures */

WIND *FIRST;
WIND *DISPLAY;
WIND *GRID;
WIND *INFORM;
WIND *LOTUS;

/*****
/* Window Messages*/
*****/

/* Messages for FIRST Window */

char title[30] = " DataFinder 5.0 Setup V. 1.0 ";
unsigned char i_bar[31] = {
    195,196,196,196,196,196,196,196,196,196,
    196,196,196,196,196,196,196,196,196,196,
    196,196,196,196,196,196,196,180 };

char item1[29] = " Display Colors ";
char item2[29] = " Video Mode - 25 Line Enable ";
char item2a[29] = " Video Mode - 43/50 Line En. ";
char item3[29] = " Cat. Auto Open - ON ";
char item3a[29] = " Cat. Auto Open - OFF ";
char item5[29] = " Quit Setup Program ";

```

6-8 Continued.

```
/* messages for DISPLAY COLORS window */

char dpitem1[29] = " Main Color Foreground ";
char dpitem2[29] = " Main Color Background ";
char dpitem3[29] = " Mouse Bar Foreground ";
char dpitem4[29] = " Mouse Bar Background ";
char dpitem5[29] = " Negative Data Foreground ";
char dpitem6[29] = " Negative Data Background ";
char dpitem7[29] = " Under Window Foreground ";
char dpitem8[29] = " Under Window Background ";
char dpitem9[29] = " Highlight Bar Foreground ";
char dpitem10[29] = " Highlight Bar Background ";
char dpitem11[29] = " Passive Wind. Bord. Fore. ";
char dpitem12[29] = " Passive Wind. Bord. Back. ";
char dpitem13[29] = " Resize Window Foreground ";
char dpitem14[29] = " Resize Window Background ";

char col1_name[8] = { "MASTERBK" };
char col2_name[13] = {
    205,205,205,'B','A','L','A','N',
    'C','E',205,205,205 };
char col3_name[19] = {
    205,205,205,205,205,
    'E','M','P','L','O','Y','E','E',
    205,205,205,205,205 };
char col1_r1[8] = " 1 ";
char col1_r2[8] = " 2 ";
char col1_r3[8] = " 3 ";
char col1_r4[8] = " 4 ";
char col2_r1[13] = " 5678.00 ";
char col2_r2[13] = " 45454.54 ";
char col2_r3[13] = " -234.44 ";
char col2_r4[13] = " 284757.55 ";
char col3_r1[19] = " Goulden ";
char col3_r2[19] = " Yung ";
char col3_r3[19] = " Fuchstein ";
char col3_r4[19] = " Chuckman ";

/* Messages for LOTUS Window */

char menu1[47] = " Mean Mode Median Range Standard Deviation ";
char mess1[47] = " Mean is the Average score of the distribution ";
char mess2[47] = " Mode is the most frequent score ";
char mess3[47] = " Median is the middle score of sample ";
char mess4[47] = " Range is the distance from highest to lowest ";
char mess5[47] = " Standard dev. is avg. distance from mean ";

/* lot_map holds mess column offset & length */

int lot_map[5][2] = {
    1,6,
    7,6,
    13,8,
    21,7,
    28,20 };

/* messages for GRID window - holds row & column */
```

```

char grid1[42] = {
    32,32,32,219,32,32,32,32,219,
    32,32,32,32,219,32,32,32,32,219,
    32,32,32,32,219,32,32,32,32,219,
    32,32,32,32,219,32,32,32,32,219,
    32,32,32 };

char grid_era[44] = {
    32,32,32,32,32,32,32,32,32,32,32,32,32,32,
    32,32,32,32,32,32,
    32,32,32,32,32,32,32,32,32,32,32,32,32,
    32,32,32,32,32,32 };

/* grid_map row,column for start of inverse item */

int grid_map[9][2] = {
    3,7,
    3,10,
    3,13,
    4,7,
    4,10,
    4,13,
    5,7,
    5,10,
    5,13 };

/* info1 window data */

unsigned char speed3[30] = { 199,196,196,196,196,196,196,196,196,
    196,196,196,196,196,196,196,196,196,
    196,196,196,196,196,196,196,196,196,
    196,196,182 };

char speed1[28] = " Press (Y) to save new ";
char speed2[28] = " Config. and quit to DOS. ";
char speed4[28] = " Press (N) to Quit to DOS ";
char speed5[28] = " and not save new Config. ";
char speed7[28] = " Press any other Key to ";
char speed8[28] = " return to DF 5.0 setup. ";

/*****/
/* global variables*/
/*****/

int xinverse;          /* attribute for inverse */
int hl_tense;          /* highlight bar intensity */

char up_dn[4] = { 32,24,25,32 };
char up_dn1[6] = { " Move " };
char up_dn2[5] = { 32,60,196,217,32 };
char up_dn3[8] = { " Select " };
char up_dn4[5] = { " ESC " };
char up_dn5[14] = { " Close Window " };
char up_dn6[29] = { " F10 - Restores Orig. Colors " };
char up_dn7[29] = { "

char dat16[4] = { 16,16,16,16 };
char dat17[4] = { 17,17,17,17 };
char dat30[4] = { 30,30,30,30 };
char dat31[4] = { 31,31,31,31 };
char dat32[4] = { 32,32,32,32 };
char dat179[4] = { 179,179,179,179 };

```

6-8 Continued.

```
char  dat186[4]  = { 186,186,186,186 };
char  dat188[4]  = { 188,188,188,188 };
char  dat191[4]  = { 191,191,191,191 };
char  dat192[4]  = { 192,192,192,192 };
char  dat196[6]  = { 196,196,196,196,196,196 };
char  dat200[4]  = { 200,200,200,200 };
char  dat203[4]  = { 203,203,203,203 };
char  dat205[28] = { 205,205,205,205,
                    205,205,205,205,
                    205,205,205,205,
                    205,205,205,205,
                    205,205,205,205,
                    205,205,205,205,
                    205,205,205,205,
                    205,205,205,205 };
char  dat217[4]  = { 217,217,217,217 };
char  dat218[4]  = { 218,218,218,218 };
char  dat254[4]  = { 254,254,254,254 };
```

```
char  dat178[80] = {
    178,178,178,178,178,178,178,178,178,178,178,
    178,178,178,178,178,178,178,178,178,178,178,
    178,178,178,178,178,178,178,178,178,178,178,
    178,178,178,178,178,178,178,178,178,178,178,
    178,178,178,178,178,178,178,178,178,178,178,
    178,178,178,178,178,178,178,178,178,178,178,
    178,178,178,178,178,178,178,178,178,178,178,
    178,178,178,178,178,178,178,178,178,178 };

```

```
/******
/*
/* INCLUDE DFINST1.C SOURCE FILE */
/*
/******

```

```
#include "dfinst1.c"
```

```
/******
/*
/* Lotus Style Window
/*
/* Receives: nothing
/* Returns: item selection number
/*
/* Displays Lotus style window
/* with attendant cursor, high-
/* light and item description
/* routines.
/*
/******

```

```
int
tlotus()
{
int key; /* scan and char value */
int exit; /* val for loop cond chk */
int exp_a; /* item explanation attr */

```



```

/*****
/* Initialize lotus menu window structure and display window */
*****/

/* Set lotus explanation Attr - Fore,Back,Intensity,Blink */
exp_a = mk_attr(MAGENTA,BLUE,ON_INTENSITY,OFF_BLINK);

/* call window initialization routines only once */
if(!lotus_flag)
{
    /* ensure window startup bypassed next window call */
    lotus_flag=1;

    /* Allocate memory and return pointer to structure */
    LOTUS = setWind(LOTUS,6,20,9,68);

    /* Set Window Attr - Fore,Back,Intensity,Blink */
    setAttr(LOTUS,mk_attr(WHITE,BLUE,ON_INTENSITY,OFF_BLINK));

    /* Set Window Border - top, bot, left, right */
    setBord(LOTUS,S_S_S_S);

    /* Set the top and bottom title - 0 set no bottom title */
    setTitle(LOTUS," Lotus Style Window ");

    /* Display window */
    strtWind(LOTUS);
}
else
    wind_display(LOTUS);

/* set loop condition */
exit=aFALSE;

do
{
    /* Write title bar - erasing old inverse */
    wind_write(LOTUS,1,1,47,menu1,LOTUS->attr);

    /* Inverse proper menu item using lot_map[][] */

    wind_attr(LOTUS,1,lot_map[lotus_item][0],lot_map[lotus_item][1],hl_tense);

    switch(lotus_item)        /* print item explanation */
    {
        case 0:

```

6-8 Continued.

```
        wind_write(LOTUS,2,1,47,mess1,exp_a);
        break;
    case 1:
        wind_write(LOTUS,2,1,47,mess2,exp_a);
        break;
    case 2:
        wind_write(LOTUS,2,1,47,mess3,exp_a);
        break;
    case 3:
        wind_write(LOTUS,2,1,47,mess4,exp_a);
        break;
    case 4:
        wind_write(LOTUS,2,1,47,mess5,exp_a);
        break;
    }

    key = kb_read();

    switch(key)
    {
        case RIGHT_ARROW: /* At right item? */
            if(lotus_item==4) /* Yes? */
                lotus_item=0; /* set left item */
            else /* Else */
                lotus_item++; /* move rt 1 item */
            break;

        case LEFT_ARROW: /* At left item? */
            if(lotus_item==0) /* Yes? */
                lotus_item=4; /* set right item */
            else /* Else */
                lotus_item--; /* move lft 1 item */
            break;

        case ENTER:
            exit=aTRUE;
            break;
    }
    } while(!exit);

/* Remove Lotus Window */

wind_remove(LOTUS);

/* return selected item number */

return(lotus_item);
}

/*****
/*
/* Grid Style Window
/*
/* Receives: nothing
/* Returns: item selection number*/
/*
/* Displays Grid style window
/* with attendant cursor & high-
/* light description routines.
/*
*****/
```

```

/*****
/* Make variables that must retain their */
/* value after the function exits global */
/*****
char top_box[3] = { 218,196,191 };
char bot_box[3] = { 192,196,217 };

void
draw_box(val)
int val;
{
int row,col;

switch(val)
{
case 0:
row=1;
col=3;
break;

case 1:
row=1;
col=8;
break;

case 2:
row=1;
col=13;
break;

case 3:
row=1;
col=18;
break;

case 4:
row=1;
col=23;
break;

case 5:
row=1;
col=28;
break;

case 6:
row=1;
col=33;
break;

case 7:
row=1;
col=38;
break;

case 8:
row=3;
col=3;

```

6-8 Continued.

```
        break;

    case 9:
        row=3;
        col=8;
        break;

    case 10:
        row=3;
        col=13;
        break;

    case 11:
        row=3;
        col=18;
        break;

    case 12:
        row=3;
        col=23;
        break;

    case 13:
        row=3;
        col=28;
        break;

    case 14:
        row=3;
        col=33;
        break;

    case 15:
        row=3;
        col=38;
        break;

    }

    wind_write(GRID,row,col,3,top_box,A13);
    wind_write(GRID,row+2,col,3,bot_box,A13);
    wind_write(GRID,row+1,col,1,dat179,A13);
    wind_write(GRID,row+1,col+2,1,dat179,A13);
}

char era_box[3] = { 32,32,32 };

void
erase_box(val)
int val;
{
    int row,col;

    switch(val)
    {
        case 0:
            row=1;
            col=3;
            break;
```

```
case 1:
    row=1;
    col=8;
    break;

case 2:
    row=1;
    col=13;
    break;

case 3:
    row=1;
    col=18;
    break;

case 4:
    row=1;
    col=23;
    break;

case 5:
    row=1;
    col=28;
    break;

case 6:
    row=1;
    col=33;
    break;

case 7:
    row=1;
    col=38;
    break;

case 8:
    row=3;
    col=3;
    break;

case 9:
    row=3;
    col=8;
    break;

case 10:
    row=3;
    col=13;
    break;

case 11:
    row=3;
    col=18;
    break;

case 12:
    row=3;
    col=23;
    break;
```

6-8 Continued.

```
case 13:
    row=3;
    col=28;
    break;

case 14:
    row=3;
    col=33;
    break;

case 15:
    row=3;
    col=38;
    break;
}
wind_write(GRID,row,col,3,era_box,GRID->attr);
wind_write(GRID,row+2,col,3,era_box,GRID->attr);
wind_write(GRID,row+1,col,1,dat32,GRID->attr);
wind_write(GRID,row+1,col+2,1,dat32,GRID->attr);
}

void
update_tta(val)
int val;
{
if(val<8)
{
    tta_fore = col_chart[val];
    ttia_fore = tta_back;
    ttia_back = tta_fore;
    tta_flag = 0;
    table_top_attr = mk_attr(tta_fore,tta_back,OFF_INTENSITY,OFF_BLINK);
}

else
{
    val -= 8;
    tta_fore = col_chart[val];
    ttia_fore = tta_back;
    ttia_back = tta_fore;
    tta_flag = 1;
    table_top_attr = mk_attr(tta_fore,tta_back,ON_INTENSITY,OFF_BLINK);
}
}

void
update_tta_back(val)
int val;
{
    tta_back = col_chart[val];
    ttia_fore = tta_back;
    ttia_back = tta_fore;
    if(!tta_flag)
        table_top_attr = mk_attr(tta_fore,tta_back,OFF_INTENSITY,OFF_BLINK);
    else
        table_top_attr = mk_attr(tta_fore,tta_back,ON_INTENSITY,OFF_BLINK);
}

void
update_mba(val)
```

```

int val;
{
if(val<8)
{
    mba_fore = col_chart[val];
    mba_flag = 0;
    mouse_bar_attr = mk_attr(mba_fore,mba_back,OFF_INTENSITY,OFF_BLINK);
}
else
{
    val -= 8;
    mba_fore = col_chart[val];
    mba_flag = 1;
    mouse_bar_attr = mk_attr(mba_fore,mba_back,ON_INTENSITY,OFF_BLINK);
}
}

void
update_mba_back(val)
int val;
{
    mba_back = col_chart[val];
    if(!mba_flag)
        mouse_bar_attr = mk_attr(mba_fore,mba_back,OFF_INTENSITY,OFF_BLINK);
    else
        mouse_bar_attr = mk_attr(mba_fore,mba_back,ON_INTENSITY,OFF_BLINK);
}

void
update_na(val)
int val;
{
    if(val<8)
    {
        na_fore = col_chart[val];
        na_flag = 0;
        negative_attr = mk_attr(na_fore,na_back,OFF_INTENSITY,OFF_BLINK);
    }
    else
    {
        val -= 8;
        na_fore = col_chart[val];
        na_flag = 1;
        negative_attr = mk_attr(na_fore,na_back,ON_INTENSITY,OFF_BLINK);
    }
}

void
update_na_back(val)
int val;
{
    na_back = col_chart[val];
    if(!na_flag)
        negative_attr = mk_attr(na_fore,na_back,OFF_INTENSITY,OFF_BLINK);
    else
        negative_attr = mk_attr(na_fore,na_back,ON_INTENSITY,OFF_BLINK);
}

void
update_wb(val)
int val;

```

6-8 Continued.

```
{
if(val<8)
{
    wb_fore = col_chart[val];
    wb_flag = 0;
    window_background = mk_attr(wb_fore,wb_back,OFF_INTENSITY,OFF_BLINK);
}
else
{
    val -= 8;
    wb_fore = col_chart[val];
    wb_flag = 1;
    window_background = mk_attr(wb_fore,wb_back,ON_INTENSITY,OFF_BLINK);
}
}

void
update_wb_back(val)
int val;
{
    wb_back = col_chart[val];
    if(!wb_flag)
        window_background = mk_attr(wb_fore,wb_back,OFF_INTENSITY,OFF_BLINK);
    else
        window_background = mk_attr(wb_fore,wb_back,ON_INTENSITY,OFF_BLINK);
}

void
update_ttia(val)
int val;
{
    if(val<8)
    {
        ttia_fore = col_chart[val];
        ttia_flag = 0;
        table_top_inv_attr = mk_attr(ttia_fore,ttia_back,OFF_INTENSITY,
        OFF_BLINK);
    }
    else
    {
        val -= 8;
        ttia_fore = col_chart[val];
        ttia_flag = 1;
        table_top_inv_attr = mk_attr(ttia_fore,ttia_back,ON_INTENSITY,
        OFF_BLINK);
    }
}

void
update_ttia_back(val)
int val;
{
    ttia_back = col_chart[val];
    if(!ttia_flag)
        table_top_inv_attr = mk_attr(ttia_fore,ttia_back,OFF_INTENSITY,
        OFF_BLINK);
    else
        table_top_inv_attr = mk_attr(ttia_fore,ttia_back,ON_INTENSITY,
        OFF_BLINK);
}
```



```

void
update_bwb(val)
int val;
{
    if(val<8)
    {
        bwb_fore = col_chart[val];
        bwb_flag = 0;
        back_wind_bord = mk_attr(bwb_fore,bwb_back,OFF_INTENSITY,OFF_BLINK);
    }
    else
    {
        val -= 8;
        bwb_fore = col_chart[val];
        bwb_flag = 1;
        back_wind_bord = mk_attr(bwb_fore,bwb_back,ON_INTENSITY,OFF_BLINK);
    }
}

void
update_bwb_back(val)
int val;
{
    bwb_back = col_chart[val];
    if(!bwb_flag)
        back_wind_bord = mk_attr(bwb_fore,bwb_back,OFF_INTENSITY,OFF_BLINK);
    else
        back_wind_bord = mk_attr(bwb_fore,bwb_back,ON_INTENSITY,OFF_BLINK);
}

void
update_ra(val)
int val;
{
    if(val<8)
    {
        ra_fore = col_chart[val];
        ra_flag = 0;
        resize_attr = mk_attr(ra_fore,ra_back,OFF_INTENSITY,OFF_BLINK);
    }
    else
    {
        val -= 8;
        ra_fore = col_chart[val];
        ra_flag = 1;
        resize_attr = mk_attr(ra_fore,ra_back,ON_INTENSITY,OFF_BLINK);
    }
}

void
update_ra_back(val)
int val;
{
    ra_back = col_chart[val];
    if(!ra_flag)
        resize_attr = mk_attr(ra_fore,ra_back,OFF_INTENSITY,OFF_BLINK);
    else
        resize_attr = mk_attr(ra_fore,ra_back,ON_INTENSITY,OFF_BLINK);
}

```

6-8 Continued.

```
/* **** */
/*
/* Simple Style Window
/*
/* Receives: nothing
/* Returns: nothing
/*
/* Displays Simple pop up
/* information window.
/*
/* **** */

/* **** */
/* Make variables that must retain their
/* value after the function exits global
/* **** */

int infol_flag=0;

int
infol()
{
int key; /* scan and char value
int red_attr;

red_attr = mk_attr(RED,WHITE,OFF_INTENSITY,OFF_BLINK);

/* **** */
/* Initialize grid menu window structure and display window
/* **** */

if(!infol_flag)
{
/* ensure window initialization bypass
infol_flag=1;

/* Allocate memory and return pointer to structure
INFORM = setWind(INFORM,12-4,20-5,22-5,49-5);

/* Set Window Attr - Fore,Back,Intensity,Blink
setAttr(INFORM,mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK));

/* Set Window Border
setBord(INFORM,D_D_D_D);

/* Set the bottom title
setTitle(INFORM," Quit ");

/* Display window
strtWind(INFORM);
}

else
wind_display(INFORM);

/* Write menu and exit messages
wind_write(INFORM,1,1,28,speed1,INFORM->attr);
wind_write(INFORM,2,1,28,speed2,INFORM->attr);
wind_write(INFORM,3,0,30,speed3,INFORM->attr);
```

```

wind_write(INFORM,4,1,28,speed4,INFORM->attr);
wind_write(INFORM,5,1,28,speed5,INFORM->attr);
wind_write(INFORM,6,0,30,speed3,INFORM->attr);
wind_write(INFORM,7,1,28,speed7,INFORM->attr);
wind_write(INFORM,8,1,28,speed8,INFORM->attr);
wind_attr(INFORM,1,10,1,red_attr);
wind_attr(INFORM,4,10,1,red_attr);

/* wait for key press */

key = kb_read();

/* remove window and display original screen information */

wind_remove(INFORM);

return (key);
}

/*
 * write_screen
 *
 * Writes the main screen
 * display.
 */

void
write_screen()
{
int attr1,attr2;
int count;
attr1 = mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK);
attr2 = mk_attr(RED,WHITE,OFF_INTENSITY,OFF_BLINK);
scrn_attr(0,0,80,attr1);
scrn_attr(24,0,80,attr2);

for(count=1; count<24; count++)
    scrn_write(count,0,80,dat178,attr1);

scrn_write(0,25,30,title,attr1);
scrn_write(24,0,4,up_dn,attr2);
scrn_write(24,4,6,up_dn1,attr1);
scrn_write(24,10,5,up_dn2,attr2);
scrn_write(24,15,8,up_dn3,attr1);
scrn_write(24,23,5,up_dn4,attr2);
scrn_write(24,28,14,up_dn5,attr1);

}

/*
 * Display Colors first level window
 */

void
display_colors()
{
int key;          /* receives Scan & char key code */
int exit;         /* holds val for main loop check */
int old_row;      /* Tracker for highlight bar */
int row;          /* Tracker for highlight bar */

```

6-8 Continued.

```
int beep;          /* flag for beep on 'Q' keypress */
int attr1,attr2;

attr1 = mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK);
attr2 = mk_attr(RED,WHITE,OFF_INTENSITY,OFF_BLINK);

/*****
/* Initialize main menu window structure and display window */
*****/

/* Allocate memory and return pointer to structure */

DISPLAY = setWind(DISPLAY,4,2,19,29);

/* Set Window Attr - Fore,Back,Intensity,Blink */

setAttr(DISPLAY,mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK));

/* Set Window Border - top, bot, left, right */

setBord(DISPLAY,S_S_S_S);

/* Set the top and bottom title */

setTitle(DISPLAY," DISPLAY COLORS ");

/* Display window */

strtWind(DISPLAY);

/* Write menu items to window */

wind_write(DISPLAY,1,1,25,dpitem1,DISPLAY->attr);
wind_write(DISPLAY,2,1,25,dpitem2,DISPLAY->attr);
wind_write(DISPLAY,3,1,25,dpitem3,DISPLAY->attr);
wind_write(DISPLAY,4,1,25,dpitem4,DISPLAY->attr);
wind_write(DISPLAY,5,1,25,dpitem5,DISPLAY->attr);
wind_write(DISPLAY,6,1,25,dpitem6,DISPLAY->attr);
wind_write(DISPLAY,7,1,25,dpitem7,DISPLAY->attr);
wind_write(DISPLAY,8,1,25,dpitem8,DISPLAY->attr);
wind_write(DISPLAY,9,1,25,dpitem9,DISPLAY->attr);
wind_write(DISPLAY,10,1,25,dpitem10,DISPLAY->attr);
wind_write(DISPLAY,11,1,25,dpitem11,DISPLAY->attr);
wind_write(DISPLAY,12,1,25,dpitem12,DISPLAY->attr);
wind_write(DISPLAY,13,1,25,dpitem13,DISPLAY->attr);
wind_write(DISPLAY,14,1,25,dpitem14,DISPLAY->attr);

/* Set highlight trackers to start at item1 (row 1) */

row = 1;
old_row = 1;

/* set default for no beep */

beep = aFALSE;

/* Set loop condition */

exit = aFALSE;
```

```

/*****
/* Main keyboard loop. Selects: tlotus(), tgrid1(),*/
/*                               info1(), & quits */
/* Up,Down arrow or First letter move highlight bar*/
*****/

do
{
    /* Write F10 message */

    scrn_write(24,45,29,up_dn6,attr1);
    scrn_attr(24,46,3,attr2);

    wind_attr(DISPLAY,old_row,1,26,DISPLAY->attr); /* off highlight bar */
    wind_attr(DISPLAY,row,1,26,xinverse);          /* on highlight bar */
    if(beep)                                         /* YES? beep after */
    {                                                /* scrn update */
        bleep();                                   /* Yes-warning beep */
        beep=aFALSE;                               /* reset-> no beep */
    }
    old_row = row;                                  /* reset OFF tracker */
    if((row==13)||(row==14))
        draw_resize();
    else
        draw_window_back();
    key = kb_read();                                /* get scan & char */
    switch(key)                                     /* eval key press */
    {
        case DOWN_ARROW:
            if(row==14)                             /* If bottom row */
                row=1;                               /* then->top row */
            else                                     /* Otherwise */
                row++;                                /* then down row */
            break;
        case UP_ARROW:
            if(row==1)                               /* If bottom row */
                row=14;                              /* then skip to 5 */
            else                                     /* Otherwise */
                row--;                                /* then up row */
            break;
        case F10:                                    /* test test test */
            dup_to_io();
            remove_df();
            draw_df();
            break;
        case ESC:
            exit=aTRUE;
            break;
        case ENTER:
            switch(row)                              /* Eval selection */
            {
                case 1:                               /* display colors */
                    scrn_write(24,45,29,up_dn7,attr1);
                    tgrid1();
                    break;

                case 2:
                    scrn_write(24,45,29,up_dn7,attr1);
                    tgrid2();
                    break;
            }
    }
}

```

6-8 Continued.

```
case 3:
    scrn_write(24,45,29,up_dn7,attr1);
    tgrid3();
    break;

case 4:
    scrn_write(24,45,29,up_dn7,attr1);
    tgrid4();
    break;

case 5:
    scrn_write(24,45,29,up_dn7,attr1);
    tgrid5();
    break;

case 6:
    scrn_write(24,45,29,up_dn7,attr1);
    tgrid6();
    break;

case 7:
    scrn_write(24,45,29,up_dn7,attr1);
    tgrid7();
    break;

case 8:
    scrn_write(24,45,29,up_dn7,attr1);
    tgrid8();
    break;

case 9:
    scrn_write(24,45,29,up_dn7,attr1);
    tgrid9();
    break;

case 10:
    scrn_write(24,45,29,up_dn7,attr1);
    tgrid10();
    break;

case 11:
    scrn_write(24,45,29,up_dn7,attr1);
    tgrid11();
    break;

case 12:
    scrn_write(24,45,29,up_dn7,attr1);
    tgrid12();
    break;

case 13:
    scrn_write(24,45,29,up_dn7,attr1);
    tgrid13();
    break;

case 14:
    scrn_write(24,45,29,up_dn7,attr1);
    tgrid14();
    break;
```

```

        }
        break;

default:                /* Check ascii val */

    key &= 0x00ff;      /* mask scan code */

    switch(key)         /* which key? */
    {
        case 'd':      /* L->lotus choice */
        case 'D':
            row=1;
            break;

        case 's':      /* G->grid choice */
        case 'S':
            row=2;
            break;

        case 'l':      /* S->simple demo */
        case 'L':
            row=3;
            break;

        case 'q':      /* Q->quit wind */
        case 'Q':
            row=5;
            beep=aTRUE; /* set for beep */
            break;
    }
    break;
}
} while (!exit);

/* remove window and restore original screen */

wind_remove(DISPLAY);
scrn_write(24,45,29,up_dn7,attr1);
}

/*****
/*
/* int main(void)
/*
/*
/* Receives: nothing
/* Returns: nothing
/*
/* Sets up the FIRST window
/* display and contains the
/* scroll bar menu selection
/* routine.
/*
/*
*****/

void
main()
{
    int key;          /* recieves Scan & char key code */
    int exit;         /* holds val for main loop check */
    int old_row;      /* Tracker for highlight bar */

```

6-8 Continued.

```
int row;          /* Tracker for highlight bar      */
int intense;      /* intensity attribute value      */
int beep;        /* flag for beep on 'Q' keypress */
int eval;

/*****/
/* Initialize VIDIO structure*/
/*                                     */
/* ALWAYS call at prog start!*/
/*****/

scrn_init();

/* Turn off the cursor */

cu_remove();

/* clear the screen */

scrn_clear();

/* write the main screen display */

write_screen();

/* fill default colors */
A1 = mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK);
A2 = mk_attr(BLUE,WHITE,OFF_INTENSITY,OFF_BLINK);
A3 = mk_attr(GREEN,WHITE,OFF_INTENSITY,OFF_BLINK);
A4 = mk_attr(CYAN,WHITE,OFF_INTENSITY,OFF_BLINK);
A5 = mk_attr(RED,WHITE,OFF_INTENSITY,OFF_BLINK);
A6 = mk_attr(MAGENTA,WHITE,OFF_INTENSITY,OFF_BLINK);
A7 = mk_attr(BROWN,WHITE,OFF_INTENSITY,OFF_BLINK);
A8 = mk_attr(WHITE,WHITE,OFF_INTENSITY,OFF_BLINK);
A9 = mk_attr(BLACK,WHITE,ON_INTENSITY,OFF_BLINK);
A10 = mk_attr(BLUE,WHITE,ON_INTENSITY,OFF_BLINK);
A11 = mk_attr(GREEN,WHITE,ON_INTENSITY,OFF_BLINK);
A12 = mk_attr(CYAN,WHITE,ON_INTENSITY,OFF_BLINK);
A13 = mk_attr(RED,WHITE,ON_INTENSITY,OFF_BLINK);
A14 = mk_attr(MAGENTA,WHITE,ON_INTENSITY,OFF_BLINK);
A15 = mk_attr(BROWN,WHITE,ON_INTENSITY,OFF_BLINK);
A16 = mk_attr(WHITE,WHITE,ON_INTENSITY,OFF_BLINK);

/* Set global attribute intense for inverse video */
xinverse = mk_attr(WHITE,BLACK,OFF_INTENSITY,OFF_BLINK);

/* set global attribute hl_tense for WHITE,WHITE,INTENSE,OFF_BLINK */
hl_tense = mk_attr(WHITE,WHITE,ON_INTENSITY,OFF_BLINK);

/* Set intense text attribute for this window */
intense = mk_attr(RED,WHITE,OFF_INTENSITY,OFF_BLINK);

/* set default data finder display colors */

table_top_attr = mk_attr(WHITE,CYAN,ON_INTENSITY,OFF_BLINK);
tta_fore = WHITE;
```



```

tta_back = CYAN;
tta_flag = 1;
table_top_inv_attr = mk_attr(WHITE,BLUE,OFF_INTENSITY,OFF_BLINK);
ttia_fore = WHITE;
ttia_back = BLUE;
ttia_flag = 0;
window_background = mk_attr(BLACK,WHITE,ON_INTENSITY,OFF_BLINK);
wb_fore = BLACK;
wb_back = WHITE;
wb_flag = 0;
data_attr = mk_attr(WHITE,CYAN,ON_INTENSITY,OFF_BLINK);
da_fore = WHITE;
da_back = CYAN;
da_flag = 1;
resize_attr = mk_attr(BLUE,CYAN,OFF_INTENSITY,OFF_BLINK);
ra_fore = BLUE;
ra_back = CYAN;
ra_flag = 0;
negative_attr = mk_attr(WHITE,RED,OFF_INTENSITY,OFF_BLINK);
na_fore = WHITE;
na_back = RED;
na_flag = 0;
mouse_bar_attr = mk_attr(WHITE,BLUE,OFF_INTENSITY,OFF_BLINK);
mba_fore = WHITE;
mba_back = BLUE;
mba_flag = 0;
back_wind_bord = mk_attr(BLACK,CYAN,OFF_INTENSITY,OFF_BLINK);
bwb_fore = BLACK;
bwb_back = CYAN;
bwb_flag = 0;

/* check to see if DF500.CGF exists */

/*****

handle = open("DF500.CFG",O_RDWR,S_IREAD|S_IWRITE);

if(handle < 0)
{
    bleep();
    bleep();
    bleep();
    write_cfg();
}
else
{
    bleep();
    read(handle,iobuff,128);
    close(handle);
    iobuff_to_data();
}

io_to_dup();

*****/

/*****/
/* Initialize main menu window structure and display window */
/*****/

/* Allocate memory and return pointer to structure */

```

6-8 Continued.

```
FIRST = setWind(FIRST,2,0,8,30);

/* Set Window Attr - Fore,Back,Intensity,Blink */
setAttr(FIRST,mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK));

/* Set Window Border - top, bot, left, right */
setBord(FIRST,S_S_S_S);

/* Set the top and bottom title */
setTitle(FIRST," MENU ");

/* Display window */
strtWind(FIRST);

/* Write menu items to window */
wind_write(FIRST,1,1,29,item1,FIRST->attr);

if(!mode_toggle)
    wind_write(FIRST,2,1,29,item2,FIRST->attr);
else
    wind_write(FIRST,2,1,29,item2a,FIRST->attr);

if(!catalog_toggle)
    wind_write(FIRST,3,1,29,item3,FIRST->attr);
else
    wind_write(FIRST,3,1,29,item3a,FIRST->attr);

wind_write(FIRST,4,0,31,i_bar,FIRST->attr);
wind_write(FIRST,5,1,29,item5,FIRST->attr);

/* Set highlight trackers to start at item1 (row 1) */

row = 1;
old_row = 1;

/* set default for no beep */
beep = aFALSE;

/* Set loop condition */
exit = aFALSE;

/*****
/* Main keyboard loop. Selects: tlotus(), tgridl(),
/* info1(), & quits */
/* Up,Down arrow or First letter move highlight bar*/
*****/

do
{
    if(!mode_toggle)
```

```

wind_write(FIRST,2,1,29,item2,FIRST->attr);

else
    wind_write(FIRST,2,1,29,item2a,FIRST->attr);

if(!catalog_toggle)
    wind_write(FIRST,3,1,29,item3,FIRST->attr);

else
    wind_write(FIRST,3,1,29,item3a,FIRST->attr);

/* highlight first letter of item */

wind_attr(FIRST,1,2,1,intense);
wind_attr(FIRST,2,2,1,intense);
wind_attr(FIRST,3,2,1,intense);
wind_attr(FIRST,5,2,1,intense);

wind_attr(FIRST,old_row,1,29,FIRST->attr); /* off highlight bar */
wind_attr(FIRST,old_row,2,1,intense);    /* intense item let */
wind_attr(FIRST,row,1,29,xinverse);      /* on highlight bar */
if(beep)                                  /* YES? beep after */
{                                          /*      scrn update */
    bleep();                             /* Yes-warning beep */
    beep=aFALSE;                         /* reset-> no beep */
}

old_row = row;                           /* reset OFF tracker */
key = kb_read();                          /* get scan & char */
switch(key)                               /* eval key press */
{
    case DOWN_ARROW:
        if(row==5)                       /* If bottom row */
            row=1;                         /* then->top row */
        else if(row==3)                   /* If row 3 */
            row=5;                         /* then skip to 5 */
        else                               /* Otherwise */
            row++;                         /* then down row */
        break;
    case UP_ARROW:
        if(row==5)                       /* If bottom row */
            row=3;                         /* then skip to 5 */
        else if(row==1)                   /* If row 3 */
            row=5;                         /* then->bot row */
        else                               /* Otherwise */
            row--;                         /* then up row */
        break;

    case ENTER:
        switch(row)                       /* Eval selection */
        {
            case 1:                       /* display colors */
                draw_df();
                display_colors();
                remove_df();
                break;

            case 2:
                if(mode_toggle==0)
                    mode_toggle=1;
                else
                    mode_toggle=0;
        }
    }
}

```

6-8 Continued.

```
        break;

    case 3:
        if(catalog_toggle==0)
            catalog_toggle=1;
        else
            catalog_toggle=0;
        break;

    case 5:
        eval = info1();
        switch(eval)
        {
            case K_Y:
            case K_y:
                write_cfg();
                exit=aTRUE;
                break;
            case K_N:
            case K_n:
                exit=aTRUE;
                break;
        }
        break;
    }
    break;

default:
    /* Check ascii val */
    key &= 0x00ff;
    /* mask scan code */
    switch(key)
    /* which key? */
    {
        case 'd':
        case 'D':
            /* L->lotus choice */
            row=1;
            break;
        case 'v':
        case 'V':
            /* G->grid choice */
            row=2;
            break;
        case 'c':
        case 'C':
            /* S->simple demo */
            row=3;
            break;
        case 'q':
        case 'Q':
            /* Q->quit wind */
            row=5;
            beep=aTRUE; /* set for beep */
            break;
    }
    break;
}
} while (!exit);

/* remove window and restore original screen */
wind_remove(FIRST);
```

```

/* turn on the cursor */

cu_display();
scrn_clear();
}

/*
 * draw data finder mini display
 */

void
draw_df()
{
draw_table_top();
draw_window_border();
draw_mouse_bar();
draw_column_name();
draw_all_data();
draw_window_back();
draw_back_wind_bord();
}

void
remove_df()
{
int attr1,count;
attr1 = mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK);
for(count=14-4; count<23; count++)
    scrn_write(count,33,44,dat178,attr1);
}

/*
 * Draw the main color portion of display
 */

void
draw_table_top()
{
scrn_write(14-4,33,44,table_top1,table_top_attr);
scrn_write(15-4,33+6,44-6,table_top2,table_top_attr);
scrn_write(15-4,33,8,table_top3,table_top_inv_attr);
draw_window_border();
draw_all_data();
scrn_write(23-4,33+11,22,grid_era,table_top_attr);
scrn_write(24-4,33+11,22,grid_era,table_top_attr);
/* scrn_write(25-4,33+11,22,grid_era,table_top_attr); */
}

void
draw_window_border()
{
int count;
scrn_write(16-4,33,44,data_border1,table_top_attr);
for(count=17-4; count<22-4; count++)
    scrn_write(count,33,1,dat186,table_top_attr);
scrn_write(22-4,33,1,dat200,table_top_attr);
scrn_write(17-4,33+43,1,dat186,table_top_attr);
scrn_write(22-4,33+43,1,dat188,table_top_attr);
scrn_write(22-4,33+42,1,dat205,table_top_attr);
}

```

6-8 Continued.

```
void
draw_all_data()
{
    scrn_write(18-4,33+1,8,col1_r1,table_top_attr);
    scrn_write(19-4,33+1,8,col1_r2,table_top_attr);
    scrn_write(20-4,33+1,8,col1_r3,table_top_attr);
    scrn_write(21-4,33+1,8,col1_r4,table_top_attr);

    scrn_write(18-4,33+10,13,col2_r1,table_top_attr);
    scrn_write(19-4,33+10,13,col2_r2,table_top_attr);
    scrn_write(20-4,33+10,13,col2_r3,negative_attr);
    scrn_write(21-4,33+10,13,col2_r4,table_top_attr);

    scrn_write(18-4,33+24,19,col3_r1,table_top_attr);
    scrn_write(19-4,33+24,19,col3_r2,table_top_attr);
    scrn_write(20-4,33+24,19,col3_r3,table_top_attr);
    scrn_write(21-4,33+24,19,col3_r3,table_top_attr);
}

void
draw_mouse_bar()
{
    int count;
    scrn_write(17-4,33+9,1,dat203,mouse_bar_attr);
    scrn_write(17-4,33+23,1,dat203,mouse_bar_attr);
    for(count=18-4; count<22-4; count++)
    {
        scrn_write(count,33+9,1,dat186,mouse_bar_attr);
        scrn_write(count,33+23,1,dat186,mouse_bar_attr);
    }
    scrn_write(22-4,33+1,1,dat17,mouse_bar_attr);
    scrn_write(22-4,33+2,39,dat178,mouse_bar_attr);
    scrn_write(22-4,33+3+38,1,dat16,mouse_bar_attr);
    scrn_write(17-4,33+43,1,dat30,mouse_bar_attr);
    scrn_write(18-4,33+43,1,dat178,mouse_bar_attr);
    scrn_write(19-4,33+43,1,dat178,mouse_bar_attr);
    scrn_write(20-4,33+43,1,dat178,mouse_bar_attr);
    scrn_write(21-4,33+43,1,dat31,mouse_bar_attr);
    draw_column_name();
}

void
draw_column_name()
{
    scrn_write(17-4,33+1,8,col1_name,mouse_bar_attr);
    scrn_write(17-4,33+10,13,col2_name,mouse_bar_attr);
    scrn_write(17-4,33+24,19,col3_name,mouse_bar_attr);
}

void
draw_window_back()
{
    scrn_write(23-4,33,10,grid_era>window_background);
    scrn_write(24-4,33,10,grid_era>window_background);
    scrn_write(25-4,33,10,grid_era>window_background);
    scrn_write(23-4,67,10,grid_era>window_background);
    scrn_write(24-4,67,10,grid_era>window_background);
    scrn_write(25-4,67,10,grid_era>window_background);
    scrn_write(26-4,33,44,grid_era>window_background);
}
```

```

void
draw_back_wind_bord()
{
    scrn_write(23-4,33+10,1,dat186,back_wind_bord);
    scrn_write(24-4,33+10,1,dat186,back_wind_bord);
    scrn_write(25-4,33+10,1,dat200,back_wind_bord);
    scrn_write(23-4,66,1,dat186,back_wind_bord);
    scrn_write(24-4,66,1,dat186,back_wind_bord);
    scrn_write(25-4,66,1,dat188,back_wind_bord);
    scrn_write(25-4,33+11,22,dat205,back_wind_bord);
}

```

```

void
draw_resize()
{
    scrn_write(19,33+1,1,dat218,resize_attr);
    scrn_write(21,33+1,1,dat192,resize_attr);
    scrn_write(19,33+7,1,dat191,resize_attr);
    scrn_write(21,33+7,1,dat217,resize_attr);
    scrn_write(19,33+2,5,dat196,resize_attr);
    scrn_write(21,33+2,5,dat196,resize_attr);
    scrn_write(20,33+1,1,dat179,resize_attr);
    scrn_write(20,33+7,1,dat179,resize_attr);
}

```

```

/*****
 * DF500.CFG file data
 * transfer routines
 */

```

```

void
data_to_iobuff()
{
    char *srce,*dest;

    iobuff[0] = (char)table_top_attr;
    iobuff[1] = (char)tta_fore;
    iobuff[2] = (char)tta_back;
    iobuff[3] = (char)tta_flag;
    iobuff[4] = (char)table_top_inv_attr;
    iobuff[5] = (char)ttia_fore;
    iobuff[6] = (char)ttia_back;
    iobuff[7] = (char)ttia_flag;
    iobuff[8] = (char>window_background;
    iobuff[9] = (char)wb_fore;
    iobuff[10] = (char)wb_back;
    iobuff[11] = (char)wb_flag;
    iobuff[12] = (char)resize_attr;
    iobuff[13] = (char)ra_fore;
    iobuff[14] = (char)ra_back;
    iobuff[15] = (char)ra_flag;
    iobuff[16] = (char)negative_attr;
    iobuff[17] = (char)na_fore;
    iobuff[18] = (char)na_back;
    iobuff[19] = (char)na_flag;
    iobuff[20] = (char)data_attr;
    iobuff[21] = (char)da_fore;
    iobuff[22] = (char)da_back;
    iobuff[23] = (char)da_flag;
}

```

6-8 Continued.

```
iobuff[24] = (char)mouse_bar_attr;
iobuff[25] = (char)mba_fore;
iobuff[26] = (char)mba_back;
iobuff[27] = (char)mba_flag;
iobuff[28] = (char)back_wind_bord;
iobuff[29] = (char)bwb_fore;
iobuff[30] = (char)bwb_back;
iobuff[31] = (char)bwb_flag;

iobuff[32] = (char)grid_item; /* item selected value */
iobuff[33] = (char)old_grid_item;
iobuff[34] = (char)grid_item1; /* item selected value */
iobuff[35] = (char)old_grid_item1;
iobuff[36] = (char)grid_item2; /* item selected value */
iobuff[37] = (char)old_grid_item2;
iobuff[38] = (char)grid_item3; /* item selected value */
iobuff[39] = (char)old_grid_item3;
iobuff[40] = (char)grid_item4; /* item selected value */
iobuff[41] = (char)old_grid_item4;
iobuff[42] = (char)grid_item5; /* item selected value */
iobuff[43] = (char)old_grid_item5;
iobuff[44] = (char)grid_item6; /* item selected value */
iobuff[45] = (char)old_grid_item6;
iobuff[46] = (char)grid_item7; /* item selected value */
iobuff[47] = (char)old_grid_item7;
iobuff[48] = (char)grid_item8; /* item selected value */
iobuff[49] = (char)old_grid_item8;
iobuff[50] = (char)grid_item9; /* item selected value */
iobuff[51] = (char)old_grid_item9;
iobuff[52] = (char)grid_item10; /* item selected value */
iobuff[53] = (char)old_grid_item10;
iobuff[54] = (char)grid_item11; /* item selected value */
iobuff[55] = (char)old_grid_item11;
iobuff[56] = (char)grid_item12; /* item selected value */
iobuff[57] = (char)old_grid_item12;
iobuff[58] = (char)grid_item13; /* item selected value */
iobuff[59] = (char)old_grid_item13;
iobuff[60] = (char)grid_item14; /* item selected value */
iobuff[61] = (char)old_grid_item14;
iobuff[62] = (char)mode_toggle;
iobuff[63] = (char)catalog_toggle;

header.main_color_attr = (char)table_top_attr;
header.main_inverse_attr = (char)table_top_inv_attr;
header.background_window_attr = (char>window_background;
header.negative_value_attr = (char)negative_attr;
header.resize_box_attr = (char)resize_attr;
header.mouse_bar_attr = (char)mouse_bar_attr;
header.passive_window_border_attr = (char)back_wind_bord;
header.mode_flag = (char)mode_toggle;
header.catalog_flag = (char)catalog_toggle;

dest = &iobuff[128];
srce = (char *) &header;
memcpy(dest,srce,sizeof(header));
}

void
iobuff_to_data()
```



```

{
table_top_attr = (int)iobuff[0];
tta_fore = (int)iobuff[1];
tta_back = (int)iobuff[2];
tta_flag = (int)iobuff[3];
table_top_inv_attr = (int)iobuff[4];
ttia_fore = (int)iobuff[5];
ttia_back = (int)iobuff[6];
ttia_flag = (int)iobuff[7];
window_background = (int)iobuff[8];
wb_fore = (int)iobuff[9];
wb_back = (int)iobuff[10];
wb_flag = (int)iobuff[11];
resize_attr = (int)iobuff[12];
ra_fore = (int)iobuff[13];
ra_back = (int)iobuff[14];
ra_flag = (int)iobuff[15];
negative_attr = (int)iobuff[16];
na_fore = (int)iobuff[17];
na_back = (int)iobuff[18];
na_flag = (int)iobuff[19];
data_attr = (int)iobuff[20];
da_fore = (int)iobuff[21];
da_back = (int)iobuff[22];
da_flag = (int)iobuff[23];
mouse_bar_attr = (int)iobuff[24];
mba_fore = (int)iobuff[25];
mba_back = (int)iobuff[26];
mba_flag = (int)iobuff[27];
back_wind_bord = (int)iobuff[28];
bwb_fore = (int)iobuff[29];
bwb_back = (int)iobuff[30];
bwb_flag = (int)iobuff[31];

grid_item = (int)iobuff[32];
old_grid_item = (int)iobuff[33];
grid_item1 = (int)iobuff[34];
old_grid_item1 = (int)iobuff[35];
grid_item2 = (int)iobuff[36];
old_grid_item2 = (int)iobuff[36];
grid_item3 = (int)iobuff[38];
old_grid_item3 = (int)iobuff[39];
grid_item4 = (int)iobuff[40];
old_grid_item4 = (int)iobuff[41];
grid_item5 = (int)iobuff[42];
old_grid_item5 = (int)iobuff[43];
grid_item6 = (int)iobuff[44];
old_grid_item6 = (int)iobuff[45];
grid_item7 = (int)iobuff[46];
old_grid_item7 = (int)iobuff[47];
grid_item8 = (int)iobuff[48];
old_grid_item8 = (int)iobuff[49];
grid_item9 = (int)iobuff[50];
old_grid_item9 = (int)iobuff[51];
grid_item10 = (int)iobuff[52];
old_grid_item10 = (int)iobuff[53];
grid_item11 = (int)iobuff[54];
old_grid_item11 = (int)iobuff[54];
grid_item12 = (int)iobuff[56];
old_grid_item12 = (int)iobuff[57];
grid_item13 = (int)iobuff[58];

```

6-8 Continued.

```
old_grid_item13 = (int)iobuff[59];
grid_item14 = (int)iobuff[60];
old_grid_item14 = (int)iobuff[61];
mode_toggle = (int)iobuff[62];
catalog_toggle = (int)iobuff[63];
}

void
write_cfg()
{
    /* create new DF500.CFG file */
    /******
    data_to_iobuff();
    handle = open("DF500.CFG",O_CREAT|O_RDWR,S_IREAD|S_IWRITE);
    write(handle,iobuff,256);
    close(handle);
    *****/
}

void
io_to_dup()
{
    char *srce,*dest;
    srce = iobuff;
    dest = dupbuff;
    memcpy(dest,srce,256);
}

void
dup_to_io()
{
    char *srce,*dest;
    srce = dupbuff;
    dest = iobuff;
    memcpy(dest,srce,256);
    iobuff_to_data();
}
```

6-8 Ends.

6-9 The source code listing to DFINST1.C.

```
/*
 * DFINST1.C
 */

/******
 * Windows for Main color set
 */

int
tgrid1()
{
    int key;    /* scan and char value */
    int exit;  /* val for loop cond chk */

    /******
    /* Initialize grid menu window structure and display window */
    /******
```

```

if(!grid_flag)
{
    /* ensure window initialization bypass */

    grid_flag=1;

    /* Allocate memory and return pointer to structure */
    GRID = setWind(GRID,2,33,8,77);

    /* Set Window Attr - Fore,Back,Intensity,Blink */
    setAttr(GRID,mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK));

    /* Set Window Border */
    setBord(GRID,D_D_D_D);

    /* Set the top and bottom title - 0 set no bottom title */
    setTitle(GRID," Change Table Top ");

    /* Display window */
    strtWind(GRID);
}

else
    wind_display(GRID);

/* Write name and exit messages */

/* set loop condition */

exit=aFALSE;

/* Write grid entries bar */

wind_write(GRID,1,1,42,grid_era,GRID->attr);
wind_write(GRID,2,1,42,grid1,GRID->attr);
wind_write(GRID,3,1,42,grid_era,GRID->attr);
wind_write(GRID,4,1,42,grid1,GRID->attr);
wind_write(GRID,5,1,42,grid_era,GRID->attr);
wind_attr(GRID,2,4,1,A1);
wind_attr(GRID,2,9,1,A2);
wind_attr(GRID,2,14,1,A3);
wind_attr(GRID,2,19,1,A4);
wind_attr(GRID,2,24,1,A5);
wind_attr(GRID,2,29,1,A6);
wind_attr(GRID,2,34,1,A7);
wind_attr(GRID,2,39,1,A8);
wind_attr(GRID,4,4,1,A9);
wind_attr(GRID,4,9,1,A10);
wind_attr(GRID,4,14,1,A11);
wind_attr(GRID,4,19,1,A12);
wind_attr(GRID,4,24,1,A13);
wind_attr(GRID,4,29,1,A14);
wind_attr(GRID,4,34,1,A15);
wind_attr(GRID,4,39,1,A16);

```

6-9 Continued.

```
do
{
erase_box(old_grid_item);
draw_box(grid_item);
old_grid_item = grid_item;
update_tta(grid_item);
draw_table_top();
key = kb_read();
switch(key)
{
case RIGHT_ARROW:

    /* IF rt col->mv to left col ELSE->mv rt */

    if(grid_item==7)
        grid_item=0;
    else if(grid_item==15)
        grid_item=8;
    else
        grid_item++;
    break;

case LEFT_ARROW:
    if(grid_item==0)
        grid_item=7;
    else if(grid_item==8)
        grid_item=15;
    else
        grid_item--;
    break;

case DOWN_ARROW:
    /* IF bottom row->mv to top row ELSE->mv down */

    if(grid_item<=7)
        grid_item += 8;
    else
        grid_item -= 8;
    break;

case UP_ARROW:
    /* IF top row->mv to bottom row ELSE->mv up */

    if(grid_item >=8)
        grid_item -= 8;
    else
        grid_item += 8;
    break;

case ENTER:
case ESC:
    exit=aTRUE;
    break;
}
} while(!exit);

/* Remove Lotus Window */

wind_remove(GRID);
```

```

/* return selected item */

return(grid_item);
}

int
tgrid2()
{
int key; /* scan and char value */
int exit; /* val for loop cond chk */

/*****
/* Initialize grid menu window structure and display window */
*****/

if(!grid_flag)
{
/* ensure window initialization bypass */

grid_flag=1;

/* Allocate memory and return pointer to structure */

GRID = setWind(GRID,2,33,8,77);

/* Set Window Attr - Fore,Back,Intensity,Blink */

setAttr(GRID,mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK));

/* Set Window Border */

setBord(GRID,D_D_D_D);

/* Set the top and bottom title - 0 set no bottom title */

setTitle(GRID," Change Table Top ");

/* Display window */

strtWind(GRID);
}

else
wind_display(GRID);

/* set loop condition */

exit=aFALSE;

/* Write grid entries bar */

wind_write(GRID,1,1,42,grid_era,GRID->attr);
wind_write(GRID,2,1,42,grid1,GRID->attr);
wind_write(GRID,3,1,42,grid_era,GRID->attr);
wind_write(GRID,4,1,42,grid_era,GRID->attr);
wind_write(GRID,5,1,42,grid_era,GRID->attr);
wind_attr(GRID,2,4,1,A1);
wind_attr(GRID,2,9,1,A2);
wind_attr(GRID,2,14,1,A3);
wind_attr(GRID,2,19,1,A4);

```

6-9 Continued.

```
wind_attr(GRID,2,24,1,A5);
wind_attr(GRID,2,29,1,A6);
wind_attr(GRID,2,34,1,A7);
wind_attr(GRID,2,39,1,A8);

do
{
    erase_box(old_grid_item1);
    draw_box(grid_item1);
    old_grid_item1 = grid_item1;
    update_tta_back(grid_item1);
    draw_table_top();
    key = kb_read();
    switch(key)
    {
        case RIGHT_ARROW:
            /* IF rt col->mv to left col ELSE->mv rt */

            if(grid_item1==7)
                grid_item1=0;
            else
                grid_item1++;
            break;

        case LEFT_ARROW:
            if(grid_item1==0)
                grid_item1=7;
            else
                grid_item1--;
            break;

        case ENTER:
        case ESC:
            exit=aTRUE;
            break;
    }
} while(!exit);

/* Remove Lotus Window */
wind_remove(GRID);

/* return selected item */
return(grid_item1);
}

/*
 * Windows for Mouse Bar
 */

int
tgrid3()
{
    int key;    /* scan and char value */
    int exit;  /* val for loop cond chk */
}
```

```

/*****
/* Initialize grid menu window structure and display window */
*****/

if(!grid_flag)
{
    /* ensure window initialization bypass */

    grid_flag=1;

    /* Allocate memory and return pointer to structure */

    GRID = setWind(GRID,2,33,8,77);

    /* Set Window Attr - Fore,Back,Intensity,Blink */

    setAttr(GRID,mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK));

    /* Set Window Border */

    setBord(GRID,D_D_D_D);

    /* Set the top and bottom title - 0 set no bottom title */

    setTitle(GRID," Change Table Top ");

    /* Display window */

    strtWind(GRID);
}

else
    wind_display(GRID);

/* set loop condition */

exit=aFALSE;

/* Write grid entries bar */

wind_write(GRID,1,1,42,grid_era,GRID->attr);
wind_write(GRID,2,1,42,grid1,GRID->attr);
wind_write(GRID,3,1,42,grid_era,GRID->attr);
wind_write(GRID,4,1,42,grid1,GRID->attr);
wind_write(GRID,5,1,42,grid_era,GRID->attr);
wind_attr(GRID,2,4,1,A1);
wind_attr(GRID,2,9,1,A2);
wind_attr(GRID,2,14,1,A3);
wind_attr(GRID,2,19,1,A4);
wind_attr(GRID,2,24,1,A5);
wind_attr(GRID,2,29,1,A6);
wind_attr(GRID,2,34,1,A7);
wind_attr(GRID,2,39,1,A8);
wind_attr(GRID,4,4,1,A9);
wind_attr(GRID,4,9,1,A10);
wind_attr(GRID,4,14,1,A11);
wind_attr(GRID,4,19,1,A12);
wind_attr(GRID,4,24,1,A13);

```

6-9 Continued.

```
wind_attr(GRID,4,29,1,A14);
wind_attr(GRID,4,34,1,A15);
wind_attr(GRID,4,39,1,A16);

do
{
    erase_box(old_grid_item2);
    draw_box(grid_item2);
    old_grid_item2 = grid_item2;
    update_mba(grid_item2);
    draw_mouse_bar();

    key = kb_read();

    switch(key)
    {
        case RIGHT_ARROW:
            /* IF rt col->mv to left col ELSE->mv rt */

            if(grid_item2==7)
                grid_item2=0;
            else if(grid_item2==15)
                grid_item2=8;
            else
                grid_item2++;
            break;

        case LEFT_ARROW:
            if(grid_item2==0)
                grid_item2=7;
            else if(grid_item2==8)
                grid_item2=15;
            else
                grid_item2--;
            break;

        case DOWN_ARROW:
            /* IF bottom row->mv to top row ELSE->mv down */

            if(grid_item2<=7)
                grid_item2 += 8;
            else
                grid_item2 -= 8;
            break;

        case UP_ARROW:
            /* IF top row->mv to bottom row ELSE->mv up */

            if(grid_item2 >=8)
                grid_item2 -= 8;
            else
                grid_item2 += 8;
            break;

        case ENTER:
        case ESC:
            exit=TRUE;
            break;
    }
}
```



```

    }
    } while(!exit);

/* Remove Lotus Window */
wind_remove(GRID);

/* return selected item */
return(grid_item2);
}

int
tgrid4()
{
int key; /* scan and char value */
int exit; /* val for loop cond chk */

/*****
/* Initialize grid menu window structure and display window */
*****/

if(!grid_flag)
{
/* ensure window initialization bypass */

grid_flag=1;

/* Allocate memory and return pointer to structure */
GRID = setWind(GRID,2,33,8,77);

/* Set Window Attr - Fore,Back,Intensity,Blink */
setAttr(GRID,mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK));

/* Set Window Border */
setBord(GRID,D_D_D_D);

/* Set the top and bottom title - 0 set no bottom title */
setTitle(GRID," Change Table Top ");

/* Display window */
strtWind(GRID);
}

else
wind_display(GRID);

/* set loop condition */
exit=aFALSE;

/* Write grid entries bar */

wind_write(GRID,1,1,42,grid_era,GRID->attr);
wind_write(GRID,2,1,42,grid1,GRID->attr);
wind_write(GRID,3,1,42,grid_era,GRID->attr);

```

6-9 Continued.

```
wind_write(GRID,4,1,42,grid_era,GRID->attr);
wind_write(GRID,5,1,42,grid_era,GRID->attr);
wind_attr(GRID,2,4,1,A1);
wind_attr(GRID,2,9,1,A2);
wind_attr(GRID,2,14,1,A3);
wind_attr(GRID,2,19,1,A4);
wind_attr(GRID,2,24,1,A5);
wind_attr(GRID,2,29,1,A6);
wind_attr(GRID,2,34,1,A7);
wind_attr(GRID,2,39,1,A8);

do
{
    erase_box(old_grid_item3);
    draw_box(grid_item3);
    old_grid_item3 = grid_item3;
    update_mba_back(grid_item3);
    draw_mouse_bar();

    key = kb_read();

    switch(key)
    {
        case RIGHT_ARROW:
            /* IF rt col->mv to left col ELSE->mv rt */

            if(grid_item3==7)
                grid_item3=0;
            else
                grid_item3++;
            break;

        case LEFT_ARROW:
            if(grid_item3==0)
                grid_item3=7;
            else
                grid_item3--;
            break;

        case ENTER:
        case ESC:
            exit=aTRUE;
            break;
    }
} while(!exit);

/* Remove Lotus Window */

wind_remove(GRID);

/* return selected item */

return(grid_item3);
}

/*
 * Windows for Mouse Bar
 */
```

```

int
tgrid5()
{
int key; /* scan and char value */
int exit; /* val for loop cond chk */

/*****
/* Initialize grid menu window structure and display window */
*****/

if(!grid_flag)
{
/* ensure window initialization bypass */

grid_flag=1;

/* Allocate memory and return pointer to structure */
GRID = setWind(GRID,2,33,8,77);

/* Set Window Attr - Fore,Back,Intensity,Blink */
setAttr(GRID,mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK));

/* Set Window Border */
setBord(GRID,D_D_D_D);

/* Set the top and bottom title - 0 set no bottom title */
setTitle(GRID," Change Table Top ");

/* Display window */
strtWind(GRID);
}

else
wind_display(GRID);

/* set loop condition */
exit=aFALSE;

/* Write grid entries bar */

wind_write(GRID,1,1,42,grid_era,GRID->attr);
wind_write(GRID,2,1,42,grid1,GRID->attr);
wind_write(GRID,3,1,42,grid_era,GRID->attr);
wind_write(GRID,4,1,42,grid1,GRID->attr);
wind_write(GRID,5,1,42,grid_era,GRID->attr);
wind_attr(GRID,2,4,1,A1);
wind_attr(GRID,2,9,1,A2);
wind_attr(GRID,2,14,1,A3);
wind_attr(GRID,2,19,1,A4);
wind_attr(GRID,2,24,1,A5);
wind_attr(GRID,2,29,1,A6);
wind_attr(GRID,2,34,1,A7);
wind_attr(GRID,2,39,1,A8);
wind_attr(GRID,4,4,1,A9);

```

6-9 Continued.

```
wind_attr(GRID,4,9,1,A10);
wind_attr(GRID,4,14,1,A11);
wind_attr(GRID,4,19,1,A12);
wind_attr(GRID,4,24,1,A13);
wind_attr(GRID,4,29,1,A14);
wind_attr(GRID,4,34,1,A15);
wind_attr(GRID,4,39,1,A16);

do
{
    erase_box(old_grid_item4);
    draw_box(grid_item4);
    old_grid_item4 = grid_item4;
    update_na(grid_item4);
    draw_all_data();

    key = kb_read();

    switch(key)
    {
        case RIGHT_ARROW:
            /* IF rt col->mv to left col ELSE->mv rt */

            if(grid_item4==7)
                grid_item4=0;
            else if(grid_item4==15)
                grid_item4=8;
            else
                grid_item4++;
            break;

        case LEFT_ARROW:
            if(grid_item4==0)
                grid_item4=7;
            else if(grid_item4==8)
                grid_item4=15;
            else
                grid_item4--;
            break;

        case DOWN_ARROW:
            /* IF bottom row->mv to top row ELSE->mv down */

            if(grid_item4<=7)
                grid_item4 += 8;
            else
                grid_item4 -= 8;
            break;

        case UP_ARROW:
            /* IF top row->mv to bottom row ELSE->mv up */

            if(grid_item4 >=8)
                grid_item4 -= 8;
            else
                grid_item4 += 8;
            break;

        case ENTER:
```

```

        case ESC:
            exit=aTRUE;
            break;
        }
    } while(!exit);

/* Remove Lotus Window */
wind_remove(GRID);

/* return selected item */
return(grid_item4);
}

int
tgrid6()
{
    int key;    /* scan and char value */
    int exit;   /* val for loop cond chk */

    /*******
    /* Initialize grid menu window structure and display window */
    /*******

    if(!grid_flag)
    {
        /* ensure window initialization bypass */

        grid_flag=1;

        /* Allocate memory and return pointer to structure */
        GRID = setWind(GRID,2,33,8,77);

        /* Set Window Attr - Fore,Back,Intensity,Blink */
        setAttr(GRID,mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK));

        /* Set Window Border */
        setBord(GRID,D_D_D_D);

        /* Set the top and bottom title - 0 set no bottom title */
        setTitle(GRID," Change Table Top ");

        /* Display window */
        strtWind(GRID);
    }

    else
        wind_display(GRID);

    /* set loop condition */
    exit=aFALSE;

    /* Write grid entries bar */

```

6-9 Continued.

```
wind_write(GRID,1,1,42,grid_era,GRID->attr);
wind_write(GRID,2,1,42,grid1,GRID->attr);
wind_write(GRID,3,1,42,grid_era,GRID->attr);
wind_write(GRID,4,1,42,grid_era,GRID->attr);
wind_write(GRID,5,1,42,grid_era,GRID->attr);
wind_attr(GRID,2,4,1,A1);
wind_attr(GRID,2,9,1,A2);
wind_attr(GRID,2,14,1,A3);
wind_attr(GRID,2,19,1,A4);
wind_attr(GRID,2,24,1,A5);
wind_attr(GRID,2,29,1,A6);
wind_attr(GRID,2,34,1,A7);
wind_attr(GRID,2,39,1,A8);

do
{
    erase_box(old_grid_item5);
    draw_box(grid_item5);
    old_grid_item5 = grid_item5;
    update_na_back(grid_item5);
    draw_all_data();

    key = kb_read();

    switch(key)
    {
        case RIGHT_ARROW:
            /* IF rt col->mv to left col ELSE->mv rt */

            if(grid_item5==7)
                grid_item5=0;
            else
                grid_item5++;
            break;

        case LEFT_ARROW:
            if(grid_item5==0)
                grid_item5=7;
            else
                grid_item5--;
            break;

        case ENTER:
        case ESC:
            exit=aTRUE;
            break;
    }
} while(!exit);

/* Remove Lotus Window */

wind_remove(GRID);

/* return selected item */

return(grid_item5);
}

/*
```

```

* Windows for Mouse Bar
*/

int
tgrid7()
{
int key; /* scan and char value */
int exit; /* val for loop cond chk */

/*****
/* Initialize grid menu window structure and display window */
*****/

if(!grid_flag)
{
/* ensure window initialization bypass */

grid_flag=1;

/* Allocate memory and return pointer to structure */
GRID = setWind(GRID,2,33,8,77);

/* Set Window Attr - Fore,Back,Intensity,Blink */
setAttr(GRID,mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK));

/* Set Window Border */
setBord(GRID,D_D_D_D);

/* Set the top and bottom title - 0 set no bottom title */
setTitle(GRID," Change Table Top ");

/* Display window */
strtWind(GRID);
}
else
wind_display(GRID);

/* set loop condition */

exit=aFALSE;

/* Write grid entries bar */

wind_write(GRID,1,1,42,grid_era,GRID->attr);
wind_write(GRID,2,1,42,grid1,GRID->attr);
wind_write(GRID,3,1,42,grid_era,GRID->attr);
wind_write(GRID,4,1,42,grid1,GRID->attr);
wind_write(GRID,5,1,42,grid_era,GRID->attr);
wind_attr(GRID,2,4,1,A1);
wind_attr(GRID,2,9,1,A2);
wind_attr(GRID,2,14,1,A3);
wind_attr(GRID,2,19,1,A4);
wind_attr(GRID,2,24,1,A5);
wind_attr(GRID,2,29,1,A6);

```

6-9 Continued.

```
wind_attr(GRID,2,34,1,A7);
wind_attr(GRID,2,39,1,A8);
wind_attr(GRID,4,4,1,A9);
wind_attr(GRID,4,9,1,A10);
wind_attr(GRID,4,14,1,A11);
wind_attr(GRID,4,19,1,A12);
wind_attr(GRID,4,24,1,A13);
wind_attr(GRID,4,29,1,A14);
wind_attr(GRID,4,34,1,A15);
wind_attr(GRID,4,39,1,A16);

do
{
    erase_box(old_grid_item6);
    draw_box(grid_item6);
    old_grid_item6 = grid_item6;
    update_wb(grid_item6);
    draw_window_back();

    key = kb_read();

    switch(key)
    {
        case RIGHT_ARROW:
            /* IF rt col->mv to left col ELSE->mv rt */

            if(grid_item6==7)
                grid_item6=0;
            else if(grid_item6==15)
                grid_item6=8;
            else
                grid_item6++;
            break;

        case LEFT_ARROW:
            if(grid_item6==0)
                grid_item6=7;
            else if(grid_item6==8)
                grid_item6=15;
            else
                grid_item6--;
            break;

        case DOWN_ARROW:
            /* IF bottom row->mv to top row ELSE->mv down */

            if(grid_item6<=7)
                grid_item6 += 8;
            else
                grid_item6 -= 8;
            break;

        case UP_ARROW:
            /* IF top row->mv to bottom row ELSE->mv up */

            if(grid_item6 >=8)
                grid_item6 -= 8;
            else
                grid_item6 += 8;
```



```

        break;

    case ENTER:
    case ESC:
        exit=aTRUE;
        break;
    }
} while(!exit);

/* Remove Lotus Window */
wind_remove(GRID);

/* return selected item */
return(grid_item6);
}

int
tgrid8()
{
    int key;    /* scan and char value */
    int exit;   /* val for loop cond chk */

    /******
    /* Initialize grid menu window structure and display window */
    /******

    if(!grid_flag)
    {
        /* ensure window initialization bypass */

        grid_flag=1;

        /* Allocate memory and return pointer to structure */

        GRID = setWind(GRID,2,33,8,77);

        /* Set Window Attr - Fore,Back,Intensity,Blink */

        setAttr(GRID,mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK));

        /* Set Window Border */

        setBord(GRID,D_D_D_D);

        /* Set the top and bottom title - 0 set no bottom title */

        setTitle(GRID," Change Table Top ");

        /* Display window */

        strtWind(GRID);
    }

    else
        wind_display(GRID);

    /* set loop condition */

    exit=aFALSE;

```

6-9 Continued.

```
/* Write grid entries bar */

wind_write(GRID,1,1,42,grid_era,GRID->attr);
wind_write(GRID,2,1,42,grid1,GRID->attr);
wind_write(GRID,3,1,42,grid_era,GRID->attr);
wind_write(GRID,4,1,42,grid_era,GRID->attr);
wind_write(GRID,5,1,42,grid_era,GRID->attr);
wind_attr(GRID,2,4,1,A1);
wind_attr(GRID,2,9,1,A2);
wind_attr(GRID,2,14,1,A3);
wind_attr(GRID,2,19,1,A4);
wind_attr(GRID,2,24,1,A5);
wind_attr(GRID,2,29,1,A6);
wind_attr(GRID,2,34,1,A7);
wind_attr(GRID,2,39,1,A8);

do
{
    erase_box(old_grid_item7);
    draw_box(grid_item7);
    old_grid_item7 = grid_item7;
    update_wb_back(grid_item7);
    draw_window_back();

    key = kb_read();

    switch(key)
    {
        case RIGHT_ARROW:
            /* IF rt col->mv to left col ELSE->mv rt */

            if(grid_item7==7)
                grid_item7=0;
            else
                grid_item7++;
            break;

        case LEFT_ARROW:
            if(grid_item7==0)
                grid_item7=7;
            else
                grid_item7--;
            break;

        case ENTER:
        case ESC:
            exit=aTRUE;
            break;
    }
} while(!exit);

/* Remove Lotus Window */

wind_remove(GRID);

/* return selected item */

return(grid_item7);
}
```

```

/*
 * Windows for Mouse Bar
 */

int
tgrid9()
{
int key; /* scan and char value */
int exit; /* val for loop cond chk */

/*****
/* Initialize grid menu window structure and display window */
*****/

if(!grid_flag)
{
/* ensure window initialization bypass */

grid_flag=1;

/* Allocate memory and return pointer to structure */

GRID = setWind(GRID,2,33,8,77);

/* Set Window Attr - Fore,Back,Intensity,Blink */

setAttr(GRID,mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK));

/* Set Window Border */

setBord(GRID,D_D_D_D);

/* Set the top and bottom title - 0 set no bottom title */

setTitle(GRID," Change Table Top ");

/* Display window */

strtWind(GRID);
}

else
wind_display(GRID);

/* set loop condition */

exit=aFALSE;

/* Write grid entries bar */

wind_write(GRID,1,1,42,grid_era,GRID->attr);
wind_write(GRID,2,1,42,gridl,GRID->attr);
wind_write(GRID,3,1,42,grid_era,GRID->attr);
wind_write(GRID,4,1,42,gridl,GRID->attr);
wind_write(GRID,5,1,42,grid_era,GRID->attr);
wind_attr(GRID,2,4,1,A1);
wind_attr(GRID,2,9,1,A2);
wind_attr(GRID,2,14,1,A3);
wind_attr(GRID,2,19,1,A4);
wind_attr(GRID,2,24,1,A5);
wind_attr(GRID,2,29,1,A6);

```

6-9 Continued.

```
wind_attr(GRID,2,34,1,A7);
wind_attr(GRID,2,39,1,A8);
wind_attr(GRID,4,4,1,A9);
wind_attr(GRID,4,9,1,A10);
wind_attr(GRID,4,14,1,A11);
wind_attr(GRID,4,19,1,A12);
wind_attr(GRID,4,24,1,A13);
wind_attr(GRID,4,29,1,A14);
wind_attr(GRID,4,34,1,A15);
wind_attr(GRID,4,39,1,A16);

do
{
    erase_box(old_grid_item8);
    draw_box(grid_item8);
    old_grid_item8 = grid_item8;
    update_ttia(grid_item8);
    draw_table_top();

    key = kb_read();

    switch(key)
    {
        case RIGHT_ARROW:
            /* IF rt col->mv to left col ELSE->mv rt */

            if(grid_item8==7)
                grid_item8=0;
            else if(grid_item8==15)
                grid_item8=8;
            else
                grid_item8++;
            break;

        case LEFT_ARROW:
            if(grid_item8==0)
                grid_item8=7;
            else if(grid_item8==8)
                grid_item8=15;
            else
                grid_item8--;
            break;

        case DOWN_ARROW:
            /* IF bottom row->mv to top row ELSE->mv down */

            if(grid_item8<=7)
                grid_item8 += 8;
            else
                grid_item8 -= 8;
            break;

        case UP_ARROW:
            /* IF top row->mv to bottom row ELSE->mv up */

            if(grid_item8 >=8)
                grid_item8 -= 8;
            else
                grid_item8 += 8;
            break;
    }
}
```

```

        case ENTER:
        case ESC:
            exit=aTRUE;
            break;
    }
} while(!exit);

/* Remove Lotus Window */
wind_remove(GRID);

/* return selected item */
return(grid_item8);
}

int
tgrid10()
{
    int key;    /* scan and char value */
    int exit;   /* val for loop cond chk */

    /***/
    /* Initialize grid menu window structure and display window */
    /***/

    if(!grid_flag)
    {
        /* ensure window initialization bypass */

        grid_flag=1;

        /* Allocate memory and return pointer to structure */

        GRID = setWind(GRID,2,33,8,77);

        /* Set Window Attr - Fore,Back,Intensity,Blink */
        setAttr(GRID,mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK));

        /* Set Window Border */
        setBord(GRID,D_D_D_D);

        /* Set the top and bottom title - 0 set no bottom title */
        setTitle(GRID," Change Table Top ");

        /* Display window */

        strtWind(GRID);
    }

    else
        wind_display(GRID);

    /* set loop condition */
    exit=aFALSE;

    /* Write grid entries bar */

```

6-9 Continued.

```
wind_write(GRID,1,1,42,grid_era,GRID->attr);
wind_write(GRID,2,1,42,grid1,GRID->attr);
wind_write(GRID,3,1,42,grid_era,GRID->attr);
wind_write(GRID,4,1,42,grid_era,GRID->attr);
wind_write(GRID,5,1,42,grid_era,GRID->attr);
wind_attr(GRID,2,4,1,A1);
wind_attr(GRID,2,9,1,A2);
wind_attr(GRID,2,14,1,A3);
wind_attr(GRID,2,19,1,A4);
wind_attr(GRID,2,24,1,A5);
wind_attr(GRID,2,29,1,A6);
wind_attr(GRID,2,34,1,A7);
wind_attr(GRID,2,39,1,A8);

do
{
    erase_box(old_grid_item9);
    draw_box(grid_item9);
    old_grid_item9 = grid_item9;
    update_ttia_back(grid_item9);
    draw_table_top();

    key = kb_read();

    switch(key)
    {
        case RIGHT_ARROW:
            /* IF rt col->mv to left col ELSE->mv rt */

            if(grid_item9==7)
                grid_item9=0;
            else
                grid_item9++;
            break;

        case LEFT_ARROW:
            if(grid_item9==0)
                grid_item9=7;
            else
                grid_item9--;
            break;

        case ENTER:
        case ESC:
            exit=aTRUE;
            break;
    }
} while(!exit);

/* Remove Lotus Window */

wind_remove(GRID);

/* return selected item */

return(grid_item9);
}
```

```

/*
 * Windows for Mouse Bar
 */

int
tgrid11()
{
int key; /* scan and char value */
int exit; /* val for loop cond chk */

/*****
/* Initialize grid menu window structure and display window */
*****/

if(!grid_flag)
{
/* ensure window initialization bypass */

grid_flag=1;

/* Allocate memory and return pointer to structure */

GRID = setWind(GRID,2,33,8,77);

/* Set Window Attr - Fore,Back,Intensity,Blink */

setAttr(GRID,mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK));

/* Set Window Border */

setBord(GRID,D_D_D_D);

/* Set the top and bottom title - 0 set no bottom title */

setTitle(GRID," Change Table Top ");

/* Display window */

strtWind(GRID);
}

else
wind_display(GRID);

/* set loop condition */

exit=aFALSE;

/* Write grid entries bar */

wind_write(GRID,1,1,42,grid_era,GRID->attr);
wind_write(GRID,2,1,42,grid1,GRID->attr);
wind_write(GRID,3,1,42,grid_era,GRID->attr);
wind_write(GRID,4,1,42,grid1,GRID->attr);
wind_write(GRID,5,1,42,grid_era,GRID->attr);
wind_attr(GRID,2,4,1,A1);
wind_attr(GRID,2,9,1,A2);
wind_attr(GRID,2,14,1,A3);
wind_attr(GRID,2,19,1,A4);
wind_attr(GRID,2,24,1,A5);

```

6-9 Continued.

```
wind_attr(GRID,2,29,1,A6);
wind_attr(GRID,2,34,1,A7);
wind_attr(GRID,2,39,1,A8);
wind_attr(GRID,4,4,1,A9);
wind_attr(GRID,4,9,1,A10);
wind_attr(GRID,4,14,1,A11);
wind_attr(GRID,4,19,1,A12);
wind_attr(GRID,4,24,1,A13);
wind_attr(GRID,4,29,1,A14);
wind_attr(GRID,4,34,1,A15);
wind_attr(GRID,4,39,1,A16);

do
{
    erase_box(old_grid_item10);
    draw_box(grid_item10);
    old_grid_item10 = grid_item10;
    update_bwb(grid_item10);
    draw_back_wind_bord();

    key = kb_read();

    switch(key)
    {
        case RIGHT_ARROW:
            /* IF rt col->mv to left col ELSE->mv rt */

            if(grid_item10==7)
                grid_item10=0;
            else if(grid_item10==15)
                grid_item10=8;
            else
                grid_item10++;
            break;

        case LEFT_ARROW:
            if(grid_item10==0)
                grid_item10=7;
            else if(grid_item10==8)
                grid_item10=15;
            else
                grid_item10--;
            break;

        case DOWN_ARROW:
            /* IF bottom row->mv to top row ELSE->mv down */

            if(grid_item10<=7)
                grid_item10 += 8;
            else
                grid_item10 -= 8;
            break;

        case UP_ARROW:
            /* IF top row->mv to bottom row ELSE->mv up */

            if(grid_item10 >=8)
                grid_item10 -= 8;
            else
```



```

        grid_item10 += 8;
        break;

        case ENTER:
        case ESC:
            exit=aTRUE;
            break;
    }
} while(!exit);

/* Remove Lotus Window */
wind_remove(GRID);

/* return selected item */
return(grid_item10);
}

int
tgrid12()
{
    int key;    /* scan and char value */
    int exit;   /* val for loop cond chk */

    /******
    /* Initialize grid menu window structure and display window */
    /******

    if(!grid_flag)
    {
        /* ensure window initialization bypass */

        grid_flag=1;

        /* Allocate memory and return pointer to structure */
        GRID = setWind(GRID,2,33,8,77);

        /* Set Window Attr - Fore,Back,Intensity,Blink */
        setAttr(GRID,mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK));

        /* Set Window Border */
        setBord(GRID,D_D_D_D);

        /* Set the top and bottom title - 0 set no bottom title */
        setTitle(GRID," Change Table Top ");

        /* Display window */
        strtWind(GRID);
    }

    else
        wind_display(GRID);

    /* set loop condition */

```

6-9 Continued.

```
exit=aFALSE;

/* Write grid entries bar */

wind_write(GRID,1,1,42,grid_era,GRID->attr);
wind_write(GRID,2,1,42,grid1,GRID->attr);
wind_write(GRID,3,1,42,grid_era,GRID->attr);
wind_write(GRID,4,1,42,grid_era,GRID->attr);
wind_write(GRID,5,1,42,grid_era,GRID->attr);
wind_attr(GRID,2,4,1,A1);
wind_attr(GRID,2,9,1,A2);
wind_attr(GRID,2,14,1,A3);
wind_attr(GRID,2,19,1,A4);
wind_attr(GRID,2,24,1,A5);
wind_attr(GRID,2,29,1,A6);
wind_attr(GRID,2,34,1,A7);
wind_attr(GRID,2,39,1,A8);

do
{
    erase_box(old_grid_item11);
    draw_box(grid_item11);
    old_grid_item11 = grid_item11;
    update_bwb_back(grid_item11);
    draw_back_wind_bord();

    key = kb_read();

    switch(key)
    {
        case RIGHT_ARROW:
            /* IF rt col->mv to left col ELSE->mv rt */

            if(grid_item11==7)
                grid_item11=0;
            else
                grid_item11++;
            break;

        case LEFT_ARROW:
            if(grid_item11==0)
                grid_item11=7;
            else
                grid_item11--;
            break;

        case ENTER:
        case ESC:
            exit=aTRUE;
            break;
    }
} while(!exit);

/* Remove Lotus Window */

wind_remove(GRID);

/* return selected item */
```

```

return(grid_item11);
}

/*
 * Windows for Mouse Bar
 */

int
tgrid13()
{
int key; /* scan and char value */
int exit; /* val for loop cond chk */

/*****
/* Initialize grid menu window structure and display window */
*****/

if(!grid_flag)
{
/* ensure window initialization bypass */

grid_flag=1;

/* Allocate memory and return pointer to structure */

GRID = setWind(GRID,2,33,8,77);

/* Set Window Attr - Fore,Back,Intensity,Blink */

setAttr(GRID,mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK));

/* Set Window Border */

setBord(GRID,D_D_D_D);

/* Set the top and bottom title - 0 set no bottom title */

setTitle(GRID," Change Table Top ");

/* Display window */

strtWind(GRID);
}

else
wind_display(GRID);

/* set loop condition */

exit=aFALSE;

/* Write grid entries bar */

wind_write(GRID,1,1,42,grid_era,GRID->attr);
wind_write(GRID,2,1,42,grid1,GRID->attr);
wind_write(GRID,3,1,42,grid_era,GRID->attr);
wind_write(GRID,4,1,42,grid1,GRID->attr);
wind_write(GRID,5,1,42,grid_era,GRID->attr);
wind_attr(GRID,2,4,1,A1);
wind_attr(GRID,2,9,1,A2);

```

6-9 Continued.

```
wind_attr(GRID,2,14,1,A3);
wind_attr(GRID,2,19,1,A4);
wind_attr(GRID,2,24,1,A5);
wind_attr(GRID,2,29,1,A6);
wind_attr(GRID,2,34,1,A7);
wind_attr(GRID,2,39,1,A8);
wind_attr(GRID,4,4,1,A9);
wind_attr(GRID,4,9,1,A10);
wind_attr(GRID,4,14,1,A11);
wind_attr(GRID,4,19,1,A12);
wind_attr(GRID,4,24,1,A13);
wind_attr(GRID,4,29,1,A14);
wind_attr(GRID,4,34,1,A15);
wind_attr(GRID,4,39,1,A16);
```

```
do
{
    erase_box(old_grid_item12);
    draw_box(grid_item12);
    old_grid_item12 = grid_item12;
    update_ra(grid_item12);
    draw_resize();

    key = kb_read();

    switch(key)
    {
        case RIGHT_ARROW:
            /* IF rt col->mv to left col ELSE->mv rt */

            if(grid_item12==7)
                grid_item12=0;
            else if(grid_item12==15)
                grid_item12=8;
            else
                grid_item12++;
            break;

        case LEFT_ARROW:
            if(grid_item12==0)
                grid_item12=7;
            else if(grid_item12==8)
                grid_item12=15;
            else
                grid_item12--;
            break;

        case DOWN_ARROW:
            /* IF bottom row->mv to top row ELSE->mv down */

            if(grid_item12<=7)
                grid_item12 += 8;
            else
                grid_item12 -= 8;
            break;

        case UP_ARROW:
            /* IF top row->mv to bottom row ELSE->mv up */
```

```

        if(grid_item12 >=8)
            grid_item12 -= 8;
        else
            grid_item12 += 8;
        break;

    case ENTER:
    case ESC:
        exit=aTRUE;
        break;
    }
} while(!exit);

/* Remove Lotus Window */
wind_remove(GRID);

/* return selected item */
return(grid_item12);
}

int
tgrid14()
{
    int key;    /* scan and char value */
    int exit;   /* val for loop cond chk */

    /***/
    /* Initialize grid menu window structure and display window */
    /***/

    if(!grid_flag)
    {
        /* ensure window initialization bypass */

        grid_flag=1;

        /* Allocate memory and return pointer to structure */
        GRID = setWind(GRID,2,33,8,77);

        /* Set Window Attr - Fore,Back,Intensity,Blink */
        setAttr(GRID,mk_attr(BLACK,WHITE,OFF_INTENSITY,OFF_BLINK));

        /* Set Window Border */
        setBord(GRID,D_D_D_D);

        /* Set the top and bottom title - 0 set no bottom title */
        setTitle(GRID," Change Table Top ");

        /* Display window */
        strtWind(GRID);
    }
}

else
    wind_display(GRID);

```

6-9 Continued.

```
/* set loop condition */

exit=aFALSE;

/* Write grid entries bar */

wind_write(GRID,1,1,42,grid_era,GRID->attr);
wind_write(GRID,2,1,42,grid1,GRID->attr);
wind_write(GRID,3,1,42,grid_era,GRID->attr);
wind_write(GRID,4,1,42,grid_era,GRID->attr);
wind_write(GRID,5,1,42,grid_era,GRID->attr);
wind_attr(GRID,2,4,1,A1);
wind_attr(GRID,2,9,1,A2);
wind_attr(GRID,2,14,1,A3);
wind_attr(GRID,2,19,1,A4);
wind_attr(GRID,2,24,1,A5);
wind_attr(GRID,2,29,1,A6);
wind_attr(GRID,2,34,1,A7);
wind_attr(GRID,2,39,1,A8);

do
{
    erase_box(old_grid_item13);
    draw_box(grid_item13);
    old_grid_item13 = grid_item13;
    update_ra_back(grid_item13);
    draw_resize();

    key = kb_read();

    switch(key)
    {
        case RIGHT_ARROW:
            /* IF rt col->mv to left col ELSE->mv rt */

            if(grid_item13==7)
                grid_item13=0;
            else
                grid_item13++;
            break;

        case LEFT_ARROW:
            if(grid_item13==0)
                grid_item13=7;
            else
                grid_item13--;
            break;

        case ENTER:
        case ESC:
            exit=aTRUE;
            break;
    }
} while(!exit);

/* Remove Lotus Window */

wind_remove(GRID);
```

```
/* return selected item */  
return(grid_item13);  
}
```

6-9 Ends.

Summary

This chapter presented many useful character mode window management functions. These functions allow you to:

- Create a window or designated size, border style, and attribute
- Write text to a window using a local coordinate system
- Write a character to the window using a local coordinate system
- Repeat a character to the window using a local coordinate system
- Read a token (8-bit char and 8-bit attribute) from a specified window location
- Change specified screen location attributes in a window
- Retrieve an alpha-numeric string from a field within a window
- Display a window
- Remove a window
- Destroy a window structure

These high-level window routines can be used to create a variety of interesting and useful user interface functions.

Chapter 7 introduces printer management functions and presents two printer management programs.

7

Printer management demonstration programs

This chapter presents two demonstration programs. The first exercises the printer management functions and the second demonstration program pulls together functions from all the categories presented in the book.

Table 7-1 presents the function prototypes for the printer related functions.

Table 7-1 Printer function prototypes.

int	print_open(int num);
int	print_close(int num);
int	print_newline(int num);
int	print_cr(int num);
int	print_string(int num, char *);
int	print_char(int num, char ch);
void	print_set_column(int num, int column);

Printer function descriptions

Function `print__open(...)`

Usage `status= print__open(num);`
 where
 `status` is an int
 `num` is an int

Remarks This function is required by the OS/2 library. The printer is treated as a device and may be written to as such. The device must be opened before it is written to. This function does not have meaning in the DOS library. The function returns a 0 on no error, and variable `num` refers to the printer port number.

Function `print__close(...)`

Usage `status= print__close(num);`
where
`status` is an int
`num` is an int

Remarks This function is required by the OS/2 library. The printer is treated as a device and may be written to as such. This device must be closed after you are finished writing to it. Closing the printer device ensures that the bytes will then be sent to the printer. This function does not have meaning in the DOS library. The function returns a 0 on no error, and variable `num` refers to the printer port number.

Function `print__newline(...)`

Usage `status= print__newline(num);`
where
`status` is an int
`num` is an int

Remarks This function sends a carriage return and line feed to the printer. Variable `status` receives a 0 on no error.

Function `print__cr(...)`

Usage `status= print__cr(num);`
where
`status` is an int
`num` is an int

Remarks This function sends a carriage return to the printer. Variable `status` receives a 0 on no error.

Function `print__string(...)`

Usage `status= print__string(num, string);`
where
`status` is an int

num is an int
string is a null terminated character buffer

Remarks This function sends a string of characters to the printer. Variable status receives a 0 on no error.

Function print__char(...)

Usage status= print__char(num, ch);
where
status is an int
num is an int
ch is a char

Remarks This function sends a byte to the printer. Variable num is the printer port and variable ch is the character that will be sent to the printer. A 0 will be returned to status on no error.

Function print__set__column(...)

Usage print__set__column(num, column);
where
num is an int
column is an int

Remarks This functions sets the column location of the print head to a specified value. Variable num holds the printer port and variable column holds the column value where the printer head will be set to.

A command line file print utility

The first demonstration program, PROG7-1.EXE, is a command line file print utility. It allows you to print Dos Text listings. Here is the program's syntax:

PROG7-1

Usage prog7-1 file__name [/d] [/c] [/p]
where
/d initiates double strike printing
/c initiates compressed printing
/p initiates header on page top with filename and page number

The options for PROG7-1 can be placed in any combination.

7-1 The source code listing to PROG7-1.C.

```
////////////////////////////////////
//
// prog7-1.c
//
// Command Line version file print
// utility
//
////////////////////////////////////

#ifdef OS2_PROG
#define INCL_DOSDEVICES
#define INCL_DOSDEVIOTCL
#define INCL_VIO
#define INCL_KBD
#define INCL_MOU
#define INCL_DOSPROCESS
#define INCL_DOSSEMAPHORES
#define FILEMAX 1000000
#else
#define FILEMAX 32000
#endif

#define CONDENSE_ON 1
#define DOUBLE_STRIKE_ON 2
#define HEADER_ON 3

#ifdef OS2_PROG
#include <os2.h>
#endif

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <memory.h>
#include <malloc.h>

#include "tproto.h"

////////////////////////////////////
//
// Epson printer defines
//
// You would put the defines for your
// printers here
//

#define EPSON_CONDENSE_ON 0x0f
#define EPSON_CONDENSE_OFF 0x12
#define EPSON_CARRIAGE_RETURN aCR
#define EPSON_LINE_FEED aLF
#define EPSON_FORM_FEED aFF
#define EPSON_DOUBLE_STRIKE_ON 0x47
#define EPSON_DOUBLE_STRIKE_OFF 0x48

////////////////////////////////////
//
```

```

// Printer defines used in PRTEXT
//

#define PRINTER_CONDENSE_ON      EPSON_CONDENSE_ON
#define PRINTER_CONDENSE_OFF    EPSON_CONDENSE_OFF
#define PRINTER_CARRIAGE_RETURN EPSON_CARRIAGE_RETURN
#define PRINTER_LINE_FEED       EPSON_LINE_FEED
#define PRINTER_FORM_FEED       EPSON_FORM_FEED
#define PRINTER_DOUBLE_STRIKE_ON EPSON_DOUBLE_STRIKE_ON
#define PRINTER_DOUBLE_STRIKE_OFF EPSON_DOUBLE_STRIKE_OFF

//
////////////////////////////////////
////////////////////////////////////
//
// Function prototypes
//

int test_arg(char *);

////////////////////////////////////
//
// Begin Program here
//

void main(int argc, char *argv[])
{
    FILE *fptr;
    ULONG chars_printed= 0;
    ULONG file_length;
    char buffer[60];
    char *file_buffer;
    char *ch, *ch_test;
    int counter, acount, ctr;
    int LF_counter= 0;
    int condense_flag= 0;
    int header_flag= 0;
    int double_strike_flag= 0;
    int page_number= 1;
    UCHAR attr1, attr2;
    int page_length= 58;
    char ascii_buffer[20];

    attr1= mk_attr(WHITE, BLUE, ON_INTENSITY, OFF_BLINK);
    attr2= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);

    file_buffer= (char *)malloc(FILEMAX);

    ch= file_buffer;

    memset(file_buffer, 0, FILEMAX);

    scrn_init();

    cu_save_loc();

    cu_remove();

    scrn_save();

```

7-1 Continued.

```
scrn_clear();

fptr= fopen(argv[1], "r");

if((argc == 1) || (!strnicmp("/", argv[1], 2))) {
    cu_move(2, 0);
    printf(" OS/2 Public Domain ASCII File Print Utility V. 1.0\n");
    printf(" Dranoel Software, xxxxxxx NY\n\n");
    printf(" Syntax: ");
    printf(" prttext file_name [/C] [/D] [/H]\n");
    printf("          where\n");
    printf(" /C Turns the Epson printer condense mode on\n");
    printf(" /D Turns the Epson printer double strike mode on\n");
    printf(" /H Prints page header with file name and page number\n");
    exit(0);
}

if(!fptr) {
    cu_move(2, 0);
    printf(" OS/2 Public Domain ASCII File Print Utility V. 1.0\n");
    printf(" Dranoel Software, xxxxxxx NY\n\n");
    printf(" File %s not found.\n\nProgram aborted.\n", argv[1]);
    exit(0);
}

scrn_change_attr(attr2);
scrn_attr(0, 0, 80, attr1);
scrn_attr(1, 0, 80, attr1);
scrn_write(0, 0, 0, " OS/2 Public Domain ASCII File Print Utility V.
1.0", attr1);
scrn_write(1, 0, 0, " Dranoel Software, xxxxxxx NY", attr1);

fread(file_buffer, FILEMAX, 1, fptr);

file_length= strlen(file_buffer);

memset(buffer, 0, 60);
sprintf(buffer, " File name: %s", argv[1]);
scrn_write(10, 0, 0, buffer, attr2);
sprintf(buffer, " File size: %d", file_length);
scrn_write(11, 0, 0, buffer, attr2);

if(argc >= 3){
    for(account= 3; account <= argc; account++) {
        switch(test_arg(argv[account-1])) {
            case CONDENSE_ON:
                condense_flag= 1;
                break;
            case DOUBLE_STRIKE_ON:
                double_strike_flag= 1;
                break;
            case HEADER_ON:
                header_flag= 1;
                break;
        }
    }
}

scrn_write(3, 0, 0, " Print Options Selected ", attr2);
```

```

scrn_repeat_char(4, 2, 23, 196, attr2);

if(condense_flag) {
    scrn_write(5,0,0," Condense Mode On", attr2);
}
else {
    scrn_write(5,0,0," Condense Mode Off", attr2);
}

if(double_strike_flag) {
    scrn_write(6,0,0," Double Strike Mode On", attr2);
}
else {
    scrn_write(6,0,0," Double Strike Mode Off", attr2);
}

if(header_flag) {
    scrn_write(7,0,0," Header Mode On (file name and page number)",
attr2);
    page_length= 55;
}
else {
    scrn_write(7,0,0," Header Mode Off", attr2);
}

print_open(0);

if(condense_flag) {
    print_char(0, PRINTER_CONDENSE_ON);
}

if(double_strike_flag) {
    print_char(0, aESC);
    print_char(0, PRINTER_DOUBLE_STRIKE_ON);
}

if(header_flag) {
    print_char(0, aCR);
    print_string(0, "File Name: ");
    print_string(0, argv[1]);
    if(condense_flag) {
        print_set_column(0, 116);
    }
    else {
        print_set_column(0, 60);
    }
    print_string(0, "Page Number: ");
    memset(ascii_buffer, 0, 4);
    itoa(page_number, ascii_buffer, 10);
    page_number++;
    print_string(0, ascii_buffer);
    print_char(0, aLF);
    print_set_column(0, 0);
    if(condense_flag) {
        for(ctr= 0; ctr < 136; ctr++) {
            print_char(0, '_');
        }
    }
    else {
        for(ctr= 0; ctr < 80; ctr++) {
            print_char(0, '_');
        }
    }
}

```

7-1 Continued.

```
    }
    }
    print_char(0, aLF);
    print_char(0, aLF);
}

////////////////////////////////////
//
// main printer loop
//

for(counter= 0; counter < file_length; counter ++){

    //////////////////////////////////
    //
    // Routine to write file name and
    // page number as header of each
    // page
    //

    if(*ch == aLF) {
        if(LF_counter >= page_length) {
            LF_counter= 0;
            print_char(0, PRINTER_FORM_FEED);
            if(header_flag) {
                print_char(0, aCR);
                print_string(0,"File Name: ");
                print_string(0, argv[1]);
                if(condense_flag) {
                    print_set_column(0, 116);
                }
                else {
                    print_set_column(0, 60);
                }
                print_string(0, "Page Number: ");
                memset(ascii_buffer, 0, 4);
                itoa(page_number, ascii_buffer, 10);
                page_number++;
                print_string(0, ascii_buffer);
                print_char(0, aLF);
                print_set_column(0, 0);
                if(condense_flag) {
                    for(ctr= 0; ctr < 136; ctr++) {
                        print_char(0, '_');
                    }
                }
                else {
                    for(ctr= 0; ctr < 80; ctr++) {
                        print_char(0, '_');
                    }
                }
                print_char(0, aLF);
                print_char(0, aLF);
            }
        }
        else {
            LF_counter++;
        }
    }
}
```



```

        print_char(0, *ch++);
        chars_printed++;
        memset(buffer, 0, 60);
        sprintf(buffer, "  Number of characters printed: %d", chars_printed);
        scrn_write(12, 0, 0, buffer, attr2);
    }

    if(double_strike_flag) {
        print_char(0, aESC);
        print_char(0, PRINTER_DOUBLE_STRIKE_OFF);
    }

    if(condense_flag) {
        print_char(0, PRINTER_CONDENSE_OFF);
    }

    print_char(0, PRINTER_FORM_FEED);

    print_close(0);

    fclose(fptr);

    free(file_buffer);

    scrn_write(14, 0, 0, "* The print job is finished.", attr2);
    scrn_attr(14, 0, 1, mk_attr(RED, WHITE, OFF_INTENSITY, ON_BLINK));
    scrn_write(15, 0, 0, "* Press any key to exit.", attr2);
    scrn_attr(15, 0, 1, mk_attr(RED, WHITE, OFF_INTENSITY, ON_BLINK));

    kb_read();

    scrn_restore();

    cu_rest_loc();

    cu_display();

}

int test_arg(char *ptr)
{
    if(!strnicmp("/C", ptr, 2)) {
        return CONDENSE_ON;
    }
    else if(!strnicmp("/D", ptr, 2)) {
        return DOUBLE_STRIKE_ON;
    }
    else if(!strnicmp("/H", ptr, 2)) {
        return HEADER_ON;
    }
    else {
        return 0;
    }
}

```

7-1 Ends.

File print utility that puts it all together

Figure 7-2 presents the listing to PROG7-2.C. This program utilizes screen, window, mouse, cursor, keyboard, make, and print functions. The program allows you to enter up to 16 files for printing. Note that the dialog boxes that were presented in Chapter 6 were used in this program.

It's plain to see that when you want to design a user interface for a program using the functions presented in this book, it makes great sense to follow the development process presented in Chapter's 6 and 7.

1. Map the dialog box using the `ms__map__display(...)` function
2. Incorporate the dialog boxes, menu bar, and drop down windows into the user interface
3. Add the code for the guts of the program

Printer demonstration program PROG7-2.C (FIG. 7-2) is not a full-fledged commercial quality print program. It will, however, provide a nice starting point for you to develop your own text file print utility program. If you do write an enhanced version of PROG7-2, know that I'd love to see it.

Examine the source code for PROG7-2.C (FIG. 7-2) very carefully. There are many useful routines that you can lift from this program, modify to your needs, and bring into your programs.

7-2 The source code listing to PROG7-2.C.

```
////////////////////////////////////
//
// prog7-2.c
//
// Demonstrates:
//   Integration of mouse and printer
//   capabilities with a menu bar/
//   drop down window user interface.
//
////////////////////////////////////

////////////////////////////////////
//
// Compiler includes here
//

#ifdef OS2_PROG
#include <os2.h>
#endif

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <memory.h>
#include <malloc.h>

////////////////////////////////////
//
// Library includes here
//
```

```

#include "tproto.h"

////////////////////////////////////
//
// Epson printer defines
//
// You would put the defines for your
// printers here
//

#define EPSON_CONDENSE_ON          0x0f
#define EPSON_CONDENSE_OFF        0x12
#define EPSON_CARRIAGE_RETURN     aCR
#define EPSON_LINE_FEED           aLF
#define EPSON_FORM_FEED           aFF
#define EPSON_DOUBLE_STRIKE_ON    0x47
#define EPSON_DOUBLE_STRIKE_OFF   0x48

////////////////////////////////////
//
// Printer defines used in PROG7-2.C
//

#define PRINTER_CONDENSE_ON        EPSON_CONDENSE_ON
#define PRINTER_CONDENSE_OFF       EPSON_CONDENSE_OFF
#define PRINTER_CARRIAGE_RETURN     EPSON_CARRIAGE_RETURN
#define PRINTER_LINE_FEED          EPSON_LINE_FEED
#define PRINTER_FORM_FEED          EPSON_FORM_FEED
#define PRINTER_DOUBLE_STRIKE_ON   EPSON_DOUBLE_STRIKE_ON
#define PRINTER_DOUBLE_STRIKE_OFF  EPSON_DOUBLE_STRIKE_OFF

//
////////////////////////////////////

////////////////////////////////////
//
// defines
//

#ifdef OS2_PROG
#define FILEMAX          1000000
#else
#define FILEMAX          32000
#endif
#define FILE_NAME_LIST_MAX 16

#define CONDENSE_ON      1
#define DOUBLE_STRIKE_ON 2
#define HEADER_ON        3

typedef struct PrintOpt {
    UCHAR    compress;
    UCHAR    double_strike;
    UCHAR    header;
    UCHAR    left_offset;
};

////////////////////////////////////
//

```

7-2 Continued.

```
// function prototypes
//
#ifdef DOS_PROG
#endif
int    openDD_File(void);
int    openDD_Print(int num_copies);
int    openDD_Options(void);
int    openDD_Help(void);
void   print_menu_bar(void);
void   FILE_About_ALT_A(void);
char   **FILE_Open_ALT_O(int *key, int *entries);
WIND   *FILE_display_files(char **files, int entries);
WIND   *PRINT_Start_ALT_S(void);

//
////////////////////////////////////
//
////////////////////////////////////
//
// global data
//

int    mouse_installed;
char   FILE_ALT_O[]= {"FILE: Open";
char   FILE_ALT_X[]= {"FILE: Exit";
char   FILE_ALT_A[]= {"FILE: About";
char   PRINT_ALT_N[]= {"PRINT: Number of copies";
char   PRINT_ALT_S[]= {"PRINT: Start print job";
char   OPTIONS_ALT_P[]= {"OPTIONS: Select printer";
char   OPTIONS_ALT_S[]= {"OPTIONS: Select print options";
char   OPTIONS_ALT_I[]= {"OPTIONS: Install printer codes";
char   HELP_ALT_P[]= {"HELP: Program help";
char   HELP_ALT_M[]= {"HELP: Dranoel help";
char   ERASE_message[]= {"";
char   dot[3]= {"['', 254, '']";
char   FILE_about_bar[30]= {"199, 196, 196, 196, 196,
                             196, 196, 196, 196, 196,
                             196, 196, 196, 196, 196,
                             196, 196, 196, 196, 196,
                             196, 196, 196, 196, 196,
                             196, 196, 196, 196, 182 };

char   FILE_about3[28]= {"Dranoel Software Inc";
char   FILE_about4[28]= {"xxxxxxxxxxxxxxxx";
char   FILE_about5[28]= {"xxxxxxxxxxxxxxxx";
char   FILE_about7[28]= {"Crafting OS/2 Utilities";
char   FILE_about8[28]= {"And Programmer Tools";
char   FILE_about9[29]= {"For Now and the Future";
char   FILE_about32[28]= {"";
char   FILE_ok[4]= {"OK";
char   FILE_open3[31]= {"Enter File Name:";
char   FILE_entry[31]= {"Is Entry Correct: [Y] [N]";
char   FILE_add[31]= {"Add Another File: [Y] [N]";
char   FILE_open32[31]= {"";
char   OPTIONS_set_bar[34]= {"199, 196, 196, 196, 196, 196,
                              196, 196, 196, 196, 196, 196,
                              196, 196, 196, 196, 196, 196,
                              196, 196, 196, 196, 196,
                              196, 196, 196, 196, 182 };
```

```

char  OPTIONS_set2[32]=  "      Options           On  Off ";
char  OPTIONS_set4[32]=  " Compress (136 dpi)   [ ] [ ] ";
char  OPTIONS_set5[32]=  " Double Strike      [ ] [ ] ";
char  OPTIONS_set6[32]=  " Header (File & Page) [ ] [ ] ";

char  OPTIONS_set8[32]=  " Left Column Offset  [00] [05] ";
char  OPTIONS_set9[32]=  "   In Blank Spaces   [10] [15] ";
char  OPTIONS_set10[32]= "                   [20] [25] ";

char  OPTIONS_set12[32]= "      [ Ok ]           [ Cancel ]      ";

//
////////////////////////////////////
////////////////////////////////////
//
// print display of file being
// printed
//

WIND *PRINT_Start_ALT_S()
{
WIND  *W1;
UCHAR  attr1, attr2;
int     key, x, y;

//
// initialize attribute
//

    attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);
    attr2= mk_attr(BLUE,  WHITE, OFF_INTENSITY, OFF_BLINK);

//
// Initialize Window
//

    W1 = wind_init(W1,
                  4,
                  0,
                  4+7,
                  39,
                  attr1,
                  D_D_D_D,
                  " Print Files ");

//
// write dialog window messages
//

    wind_write(W1, 2, 1, 0, " File Name: ", attr2);
    wind_write(W1, 3, 1, 0, " File Size: ", attr2);
    wind_write(W1, 4, 1, 0, " Number of Chars Printed: ", attr2);

return(W1);
}

```

7-2 Continued.

```
//
////////////////////////////////////

////////////////////////////////////
//
// display file names
//

WIND *FILE_display_files(char **files, int num_entries)
{
    WIND    *W1;
    UCHAR   attr1;
    int     key, counter;

    //
    // initialize attribute
    //

    attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);

    //
    // Initialize Window
    //

    W1 = wind_init(W1,
                  2,
                  40,
                  22,
                  79,
                  attr1,
                  D_D_D_D,
                  " File Print List ");

    //
    // write dialog window messages
    //

    wind_write(W1, 2, 2, 9, "File Name", W1->attr);
    wind_repeat_char(W1, 3, 2, 9, 196, W1->attr);
    wind_write(W1, 2, 26, 12, "Print Status", W1->attr);
    wind_repeat_char(W1, 3, 26, 12, 196, W1->attr);

    for(counter= 0; counter < num_entries; counter++) {
        wind_write(W1, 4 + counter, 2, 0, (char *) *(files + (counter *
sizeof(char *))), W1->attr);
        wind_write(W1, 4 + counter, 26, 0, "Waiting...", W1->attr);
    }

    return(W1);
}
```

```

////////////////////////////////////
//
// OPTIONS: Set Print Options

int OPTIONS_Set_ALT_S(struct PrintOpt *P01)
{
WIND    *W1;
UCHAR   attr1, attr2, compress, double_strike, change= 0;
UCHAR   header, left_offset, exit_flag= 0;
int      x, y, key;

//
// transfer PrintOpt structure values to
// local variables
//

    compress= P01->compress;
    double_strike= P01->double_strike;
    header= P01->header;
    left_offset= P01->left_offset;

//
// initialize attribute
//

    attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);
    attr2= mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK);

//
// turn off the mouse
//

    if(mouse_installed) {
        ms_off();
    }

//
// Initialize Window
//

    W1 = wind_init(W1,
                    4,
                    20,
                    4+13,
                    20+33,
                    attr1,
                    D_D_D_D,
                    " Print Options ");

//
// write dialog window messages
//

    wind_write(W1, 0, 1, 3, dot, W1->attr);

    wind_write(W1, 2, 1, 32, OPTIONS_set2, W1->attr);
    wind_repeat_char(W1, 3, 7, 7, 196, W1->attr);
    wind_repeat_char(W1, 3, 25, 3, 196, W1->attr);
    wind_repeat_char(W1, 3, 29, 3, 196, W1->attr);
    wind_write(W1, 4, 1, 32, OPTIONS_set4, W1->attr);
    wind_char(W1, 4, 2, 'C', attr2);
    wind_write(W1, 5, 1, 32, OPTIONS_set5, W1->attr);

```

7-2 Continued.

```
wind_char(W1, 5, 2, 'D', attr2);
wind_write(W1, 6, 1, 32, OPTIONS_set6, W1->attr);
wind_char(W1, 6, 2, 'H', attr2);
wind_write(W1, 7, 1, 32, OPTIONS_set8, W1->attr);
wind_char(W1, 7, 2, 'L', attr2);
wind_write(W1, 8, 1, 32, OPTIONS_set9, W1->attr);
wind_write(W1, 9, 1, 32, OPTIONS_set10, W1->attr);
wind_write(W1, 11, 1, 32, OPTIONS_set12, W1->attr);

//
// display current setup
//

wind_char(W1, 11, 8, 'O', attr2);
wind_char(W1, 11, 22, 'C', attr2);

if(compress) {
    wind_char(W1, 4, 26, 'X', attr2);
}
else {
    wind_char(W1, 4, 30, 'X', attr2);
}

if(double_strike) {
    wind_char(W1, 5, 26, 'X', attr2);
}
else {
    wind_char(W1, 5, 30, 'X', attr2);
}

if(header) {
    wind_char(W1, 6, 26, 'X', attr2);
}
else {
    wind_char(W1, 6, 30, 'X', attr2);
}

switch(left_offset) {
    case 00:
        wind_attr(W1, 7, 24, 2, attr2);
        break;
    case 10:
        wind_attr(W1, 8, 24, 2, attr2);
        break;
    case 20:
        wind_attr(W1, 9, 24, 2, attr2);
        break;
    case 05:
        wind_attr(W1, 7, 29, 2, attr2);
        break;
    case 15:
        wind_attr(W1, 8, 29, 2, attr2);
        break;
    case 25:
        wind_attr(W1, 9, 29, 2, attr2);
        break;
}
```



```

//
// turn on the mouse
//

    if(mouse_installed) {
        ms_on();
    }

// ms_map_display(20, 0, F10);

//
// process keyboard and mouse input
//

    exit_flag= 0;

    do {
        if(change) {
            change= 0;

            if(mouse_installed) {
                ms_off();
            }

            if(compress) {
                wind_char(W1, 4, 26, 'X', attr2);
                wind_char(W1, 4, 30, ' ', W1->attr);
            }
            else {
                wind_char(W1, 4, 26, ' ', W1->attr);
                wind_char(W1, 4, 30, 'X', attr2);
            }

            if(double_strike) {
                wind_char(W1, 5, 26, 'X', attr2);
                wind_char(W1, 5, 30, ' ', W1->attr);
            }
            else {
                wind_char(W1, 5, 26, ' ', W1->attr);
                wind_char(W1, 5, 30, 'X', attr2);
            }

            if(header) {
                wind_char(W1, 6, 26, 'X', attr2);
                wind_char(W1, 6, 30, ' ', W1->attr);
            }
            else {
                wind_char(W1, 6, 26, ' ', W1->attr);
                wind_char(W1, 6, 30, 'X', attr2);
            }

            wind_attr(W1, 7, 24, 2, W1->attr);
            wind_attr(W1, 8, 24, 2, W1->attr);
            wind_attr(W1, 9, 24, 2, W1->attr);
            wind_attr(W1, 7, 29, 2, W1->attr);
            wind_attr(W1, 8, 29, 2, W1->attr);
            wind_attr(W1, 9, 29, 2, W1->attr);

            switch(left_offset) {
                case 00:
                    wind_attr(W1, 7, 24, 2, attr2);

```

7-2 Continued.

```
        break;
    case 10:
        wind_attr(W1, 8, 24, 2, attr2);
        break;
    case 20:
        wind_attr(W1, 9, 24, 2, attr2);
        break;
    case 05:
        wind_attr(W1, 7, 29, 2, attr2);
        break;
    case 15:
        wind_attr(W1, 8, 29, 2, attr2);
        break;
    case 25:
        wind_attr(W1, 9, 29, 2, attr2);
        break;
    }
    if(mouse_installed) {
        ms_on();
    }
}

key= kb_status();

if(!key) && (mouse_installed)) {
    key= ms_status(&x, &y);

    if((key == 1) && (x >= 208) && (x <= 248) && (y == 120)) {
        key= ENTER;
    }

    if(((key == 1) && (x >= 320) && (x <= 392) && (y == 120)) ||
        ((x == 176) && (y == 32))) {
        key= ESCAPE;
    }

    // Compress off to on

    if((key == 1) && (x == 368) && (y == 64)) {
        if(!compress) {
            key= ALT_C;
        }
    }

    // Compress on to off

    if((key == 1) && (x == 400) && (y == 64)) {
        if(compress) {
            key= ALT_C;
        }
    }

    // Double strike off to on

    if((key == 1) && (x == 368) && (y == 72)) {
        if(!double_strike) {
            key= ALT_D;
        }
    }
}
```

```

// Double strike on to off
if((key == 1) && (x == 400) && (y == 72)) {
    if(double_strike) {
        key= ALT_D;
    }
}

// header off to on
if((key == 1) && (x == 368) && (y == 80)) {
    if(!header) {
        key= ALT_H;
    }
}

// header on to off
if((key == 1) && (x == 400) && (y == 80)) {
    if(header) {
        key= ALT_H;
    }
}

if((key == 1) && (x >= 352) && (x <= 360) && (y == 88)) {
    left_offset= 00;
    change= 1;
}
if((key == 1) && (x >= 392) && (x <= 400) && (y == 88)) {
    left_offset= 05;
    change= 1;
}
if((key == 1) && (x >= 352) && (x <= 360) && (y == 96)) {
    left_offset= 10;
    change= 1;
}
if((key == 1) && (x >= 392) && (x <= 400) && (y == 96)) {
    left_offset= 15;
    change= 1;
}
if((key == 1) && (x >= 352) && (x <= 360) && (y == 104)) {
    left_offset= 20;
    change= 1;
}
if((key == 1) && (x >= 392) && (x <= 400) && (y == 104)) {
    left_offset= 25;
    change= 1;
}
}

if(key == ALT_C) {
    if(!compress) {
        compress= 1;
    }
    else {
        compress= 0;
    }
    change= 1;
}

if(key == ALT_D) {

```

7-2 Continued.

```
        if(!double_strike) {
            double_strike= 1;
        }
        else {
            double_strike= 0;
        }
        change= 1;
    }

    if(key == ALT_L) {
        if(left_offset == 25) {
            left_offset= 0;
        }
        else {
            left_offset+= 5;
        }
        change= 1;
    }

    if(key == ALT_H) {
        if(!header) {
            header= 1;
        }
        else {
            header= 0;
        }
        change= 1;
    }

    if((key == ESCAPE) || (key == ENTER)){
        exit_flag= 1;
    }

    } while(!exit_flag);

//
// if key is equal to ENTER then
// transfer the data from the
// local shadow of the
// PrintOpt structure to the
// parameter PrintOpt pointer's
// structure
//
    if(key == ENTER) {
        P01->compress= compress;
        P01->double_strike= double_strike;
        P01->header= header;
        P01->left_offset= left_offset;
    }

//
// turn off the mouse
//
    if(mouse_installed) {
        ms_off();
    }
}
```

```

//
// remove window and display original screen information
//

    wind_remove(W1);

//
// destroy the window structure
//

    wind_destroy(W1);

//
// turn on the mouse
//

    if(mouse_installed) {
        ms_on();
    }

return key;
}

////////////////////////////////////
//
// FILE: Open dialog box

char **FILE_Open_ALT_0(int *key_val, int *entries)
{
WIND    *W1;
FILE    *fptr;
UCHAR   attr1;
UCHAR   exit_flag= 0, exit_flag1= 0;
char     *cptr, **cptrptr;
int      key;
int      x;
int      y;
int      file_count= 0;
int      count= 0;

//
// initialize array of pointers
//

    cptrptr= (char **)calloc(FILE_NAME_LIST_MAX, sizeof(char *));

//
// initialize the array of pointers to 0
//

    for(count= 0; count < FILE_NAME_LIST_MAX; count++) {
        *(cptrptr + (count * sizeof(char *)))= (char *)0;
    }

//
// initialize attribute

```

7-2 Continued.

```
//  
  
    attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);  
  
//  
// turn off the mouse  
//  
  
    if(mouse_installed) {  
        ms_off();  
    }  
  
//  
// Initialize Window  
//  
  
    W1 = wind_init(W1,  
                  8,  
                  8,  
                  4+11,  
                  72,  
                  attr1,  
                  D_D_D_D,  
                  " Open File For Print ");  
  
//  
// write dialog window messages  
//  
  
    wind_write(W1, 0, 1, 3, dot, W1->attr);  
    wind_write(W1, 1, 1, 31, FILE_open32, W1->attr);  
    wind_write(W1, 2, 1, 31, FILE_open3, W1->attr);  
  
//  
// turn on the mouse  
//  
  
    if(mouse_installed) {  
        ms_on();  
    }  
  
//  ms_map_display(3, 0, F10);  
  
//  
// read the keyboard  
//  
  
    do {  
        // jump her for new entry  
  
        add_entry:  
  
        // allocate memory for file name  
  
        cptr= (char *)malloc(40);  
  
        // place pointer in array of pointers
```

```

*(cptrptr + (file_count * sizeof(char *)))= cptr;

// initialize file name buffer to 0
memset(cptr, 0, 40);

// jump here on entry abort
abort_entry:

    // display mouse
    cu_display();

    // if the mouse is installed turn it off
    if(mouse_installed) {
        ms_off();
    }

    // get file name string from keyboard
    key= wind_kb_edit(W1,
                     cptr,
                     2,
                     19,
                     34,
                     UPPER,
                     W1->attr);
    // remove the mouse
    cu_remove();

    // if the key is ENTER process name
    if(key == ENTER) {
        // erase the file name from enter row
        wind_repeat_char(W1, 2, 19, 34, ' ', W1->attr);

        // copy the file name to the line below
        wind_write(W1, 3, 2, 0, "File Entry: ", W1->attr);
        wind_write(W1, 3, 14, 0, cptr, W1->attr);

        // test to see if the file can be opened
        fptr= fopen(cptr, "r");

        // on no print error message and allow user to
        // enter another name
        if(!fptr) {
            wind_write(W1,
                      4,
                      2,
                      0,
                      "Error: File Not Found - ESC to continue",
                      mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK));

            // wait for key press

```

7-2 Continued.

```
        kb_read();

        // erase error messages

        wind_repeat_char(W1, 3, 2, 50, ' ', W1->attr);
        wind_repeat_char(W1, 4, 2, 50, ' ', W1->attr);
        fclose(fp);
        goto abort_entry;
    }

    // close the file

    else {
        fclose(fp);
    }

    // write the file entry string to the screen

    wind_write(W1, 4, 1, 31, FILE_entry, W1->attr);
}

// abort file entry process when wind_kb_edit
// return any key other than ENTER

else {
    goto abort_file_entry_process;
}

// if the mouse is installed then turn it on

if(mouse_installed) {
    ms_on();
}

// initialize loop exit flag

exit_flag1= 0;

// loop for key or mouse response to file name

do {

    // no wait and get key press

    key= kb_status();

    // if no key press and the mouse is installed

    if((!key) && (mouse_installed)) {

        // get the mouse status

        key= ms_status(&x, &y);

        // mouse press on Y

        if((key == 1) && (x == 264) && (y == 96)) {
            key= K_Y;
        }
    }
}
```



```

        // mouse press on N
        if((key == 1) && (x == 296) && (y == 96)) {
            key= K_N;
        }

        // mouse press on close window button
        if((key == 1) && (x == 80) && (y == 64)) {
            key= ESCAPE;
        }

    }

    // ENTER and Y evoke same action
    if(key == ENTER) {
        key= K_Y;
    }

    // ESCAPE aborts process
    if(key == ESCAPE) {
        goto abort_file_entry_process;
    }

    // allow for either lower case or upper case entry
    if((key == K_Y) || (key == K_y)) {
        key= K_Y;
    }
    if((key == K_N) || (key == K_n)) {
        key= K_N;
    }
    if((key == K_Y) || (key == K_N)) {
        exit_flag1= 1;
    }

    // loop until exit flag is set
    } while(!exit_flag1);

// if the mouse is installed turn it off
if(mouse_installed) {
    ms_off();
}

// erase screen messages
wind_write(W1, 2, 19, 31, FILE_open32, W1->attr);
wind_write(W1, 3, 1, 31, FILE_open32, W1->attr);

// abort process on N
if(key == K_N) {
    wind_write(W1, 3, 1, 31, FILE_open32, W1->attr);
    wind_write(W1, 4, 1, 31, FILE_open32, W1->attr);
    goto abort_entry;
}

```

7-2 Continued.

```
// file add process

save_or_not:

wind_write(W1, 4, 1, 31, FILE_add, W1->attr);

// if the mouse is installed then turn it on

if(mouse_installed) {
    ms_on();
}

// initialize the exit flag

exit_flag1= 0;

do {

    // no wait and get key press

    key= kb_status();

    // if no key press and the mouse is installed

    if((!key) && (mouse_installed)) {

        // get the mouse status

        key= ms_status(&x, &y);

        // mouse press on Y

        if((key == 1) && (x == 264) && (y == 96)) {
            key= K_Y;
        }

        // mouse press on N

        if((key == 1) && (x == 296) && (y == 96)) {
            key= K_N;
        }

        // mouse press on close window button

        if((key == 1) && (x == 80) && (y == 64)) {
            key= ESCAPE;
        }

    }

    // ENTER and Y evoke same action

    if(key == ENTER) {
        key= K_Y;
    }

    // ESCAPE aborts process

    if(key == ESCAPE) {
        goto abort_file_entry_process;
    }
}
```

```

    }

    // allow for either lower case or upper case entry
    if((key == K_Y) || (key == K_y)) {
        key= K_Y;
    }
    if((key == K_N) || (key == K_n)) {
        key= K_N;
    }
    if((key == K_Y) || (key == K_N)) {
        exit_flag1= 1;
    }

    // loop until exit flag is set
    } while(!exit_flag1);

// if the mouse is installed turn it off
if(mouse_installed) {
    ms_off();
}

// on yes process entry and adjust file counter
if( key == K_Y) {
    if(file_count == 14) {
        wind_write(W1,
                    4,
                    2,
                    0,
                    "File Limit Reached - ESC to continue",
                    mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK));
        // wait for key press

        kb_read();

        // erase error messages

        wind_repeat_char(W1, 3, 2, 50, ' ', W1->attr);
        wind_repeat_char(W1, 4, 2, 50, ' ', W1->attr);

        goto save_or_not;
    }

    wind_write(W1, 3, 1, 31, FILE_open32, W1->attr);
    wind_write(W1, 4, 1, 31, FILE_open32, W1->attr);
    file_count++;
    goto add_entry;
}

// if the mouse is installed turn it on
if(mouse_installed) {
    ms_on();
}

// initialize exit flag

```

7-2 Continued.

```
    exit_flag= 1;

    } while(!exit_flag);

//
// jump here on abort of the file entry process
//

    abort_file_entry_process:

    *entries= file_count + 1;

//
// return key_val to calling function
//

    *key_val= key;

//
// turn off the mouse
//

    if(mouse_installed) {
        ms_off();
    }

//
// remove window and display original screen information
//

    wind_remove((WIND *)W1);

//
// destroy the window structure
//

    wind_destroy((WIND *)W1);

//
// turn on the mouse
//

    if(mouse_installed) {
        ms_on();
    }

return cptrptr;
}

//
////////////////////////////////////

////////////////////////////////////
//
// FILE: About dialog box

void
```

```

FILE_About_ALT_A()
{
WIND    *W1;
UCHAR   attr1, attr2, exit_flag= 0;
int      key= 0, x= 0, y= 0;

//
// initialize attribute
//

    attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);
    attr2= mk_attr(BLUE, WHITE, OFF_INTENSITY, OFF_BLINK);

//
// turn off the mouse
//

    if(mouse_installed) {
        ms_off();
    }

//
// Initialize Window
//

    W1 = wind_init(W1,
                    4,
                    24,
                    4+14,
                    24+29,
                    attr1,
                    D_D_D_D,
                    "");

//
// write dialog window messages
//

    wind_write(W1, 0, 1, 3, dot, W1->attr);
    wind_write(W1, 1, 1, 28, FILE_about32, W1->attr);
    wind_write(W1, 2, 1, 28, FILE_about3, attr2);
    wind_write(W1, 3, 1, 28, FILE_about4, attr2);
    wind_write(W1, 4, 1, 28, FILE_about5, attr2);
    wind_write(W1, 5, 1, 28, FILE_about32, W1->attr);
    wind_write(W1, 6, 0, 30, FILE_about_bar, W1->attr);
    wind_write(W1, 7, 1, 28, FILE_about7, attr2);
    wind_write(W1, 8, 1, 28, FILE_about8, attr2);
    wind_write(W1, 9, 1, 28, FILE_about9, attr2);
    wind_write(W1, 10, 0, 30, FILE_about_bar, W1->attr);
    wind_write(W1, 11, 1, 28, FILE_about32, W1->attr);
    wind_write(W1, 12, 13, 4, FILE_ok, mk_attr_inverse(attr2));

//
// turn on the mouse
//

    if(mouse_installed) {
        ms_on();
    }
}

```

7-2 Continued.

```
//  
// read the keyboard  
//  
do {  
    // check to see if key press  
    key= kb_status();  
    // if no key press then read the mouse  
    if((!key) && (mouse_installed)) {  
        // check for button press  
        key= ms_status(&x, &y);  
        // if button press and mouse on the dot  
        if((key == 1) && (x == 208) && (y == 32)) {  
            key= ENTER;  
        }  
        // if button press and mouse on OK  
        if((key == 1) && (x >= 296) && (x <= 320) && (y == 128)) {  
            key= ENTER;  
        }  
    }  
    if(key == ENTER) {  
        exit_flag= 1;  
    }  
    } while(!exit_flag);  
  
//  
// turn off the mouse  
//  
    if(mouse_installed) {  
        ms_off();  
    }  
  
//  
// remove window and display original screen information  
//  
    wind_remove(W1);  
  
//  
// destroy the window structure  
//  
    wind_destroy(W1);  
  
//  
// turn on the mouse  
//
```

```

        if(mouse_installed) {
            ms_on();
        }
    }

    //////////////////////////////////////
    //
    // print the menu bar
    //

    void print_menu_bar()
    {
        UCHAR attr1, attr2;

        //
        // make the attributes
        //

        attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);
        attr2= mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK);

        //
        // print menu bar
        //

        scrn_attr(0, 0, 80, attr1);

        scrn_write(0,
                   0,
                   0,
                   " File Print Options Help",
                   attr1);

        //
        // highlight hot keys
        //

        scrn_attr(0, 1, 1, attr2);           // highlight F
        scrn_attr(0, 7, 1, attr2);           // highlight E
        scrn_attr(0, 14, 1, attr2);          // highlight O
        scrn_attr(0, 23, 1, attr2);          // highlight H

        //
        // turn on mouse if installed
        //

        if(mouse_installed) {
            ms_on();
        }
    }

    //
    //////////////////////////////////////
    //////////////////////////////////////
    //
    // open File drop down window
    //

```

7-2 Continued.

```
int openDD_File()
{
    UCHAR attr1, attr2, exit_flag= 0;
    char split_items[12]= { 195, 196, 196, 196, 196, 196,
                           196, 196, 196, 196, 196, 180 };

    RECT *R;
    int key, oldrow= 0, newrow= 0;
    int x, y;

    //
    // set attributes
    //

    attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);
    attr2= mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK);

    //
    // initialize the rectangular structure
    //

    R= setRect(R,
               1,
               0,
               1 + 5,
               1 + 10);

    //
    // save the screen image under the rectangle
    //

    saveRect(R);

    //
    // draw a rectangular box under the File option
    // using a Single sided border (S_S_S_S);
    //

    boxRect(R,
            S_S_S_S,
            attr1);

    //
    // attributes change on File option
    //

    scrn_attr(0, 0, 6, attr1);

    // write the menu items to the screen

    scrn_write(2,
               1,
               0,
               " Open      ",
               attr1);

    scrn_write(3,
               1,
               0,
               " Exit      ",
               attr1);
```



```

scrn_write(4,
           0,
           12,
           split_items,
           attr1);

scrn_write(5,
           1,
           0,
           " About... ",
           attr1);

scrn_attr(oldrow + 2, 2, 8, attr1);

scrn_attr(2,
          2,
          1,
          attr2);

scrn_attr(3,
          3,
          1,
          attr2);

scrn_attr(5,
          2,
          1,
          attr2);

// draw highlight bar
scrn_attr(newrow + 2, 1, 10, mk_attr_inverse(attr1));

if(mouse_installed) {
    ms_on();
}
//
// main keyboard loop
//

do {

    if(oldrow != newrow) {

        if(mouse_installed) {
            ms_off();
        }

        // write the menu items to the screen

        scrn_write(2,
                   1,
                   0,
                   " Open      ",
                   attr1);

        scrn_write(3,
                   1,
                   0,
                   " Exit      ",
                   attr1);
    }
}

```

7-2 Continued.

```
        scrn_write(4,
                    0,
                    12,
                    split_items,
                    attr1);

        scrn_write(5,
                    1,
                    0,
                    " About... ",
                    attr1);

        scrn_attr(oldrow + 2, 2, 8, attr1);

        scrn_attr(2,
                    2,
                    1,
                    attr2);

        scrn_attr(3,
                    3,
                    1,
                    attr2);

        scrn_attr(5,
                    2,
                    1,
                    attr2);

        // draw highlight bar

        scrn_attr(newrow + 2, 1, 10, mk_attr_inverse(attr1));

        oldrow= newrow;

        // if mouse installed turn on the mouse

        if(mouse_installed) {
            ms_on();
        }

    }

    key= kb_status();

    if((!key) && (mouse_installed)) {

        key= ms_status(&x, &y);

        if((key == 1) && (x >= 8) && (x <= (48 + 32)) && (y == 16)) {
            key= ALT_0;
        }

        if((key == 1) && (x >= 8) && (x <= (48 + 32)) && (y == 24)) {
            key= ALT_X;
        }

        if((key == 1) && (x >= 8) && (x <= (48 + 32)) && (y == 40)) {
            key= ALT_A;
        }
    }
```

```

        if((key == 1) && (y >= 8)) {
            key= ESCAPE;
        }

        if((key == 2) && (y >= 8)) {
            key= RIGHT_ARROW;
        }
    }

    // process key press

switch(key) {
    // Item selected so
    // break from the loop

    case ENTER:
        if(newrow == 1) {
            key= ALT_X;
        }
        else {
            key= ALT_0;
        }
        exit_flag= 1;
        break;

    // select file for printing
    // quit program
    // no action

    case RIGHT_ARROW:
    case RIGHT_ARROW_K:
    case LEFT_ARROW:
    case LEFT_ARROW_K:
    case ALT_A:
    case ALT_0:
    case ALT_X:
    case ESCAPE:
        exit_flag= 1;
        break;

    // move highlight bar down

    case DOWN_ARROW:
    case DOWN_ARROW_K:
        if(newrow == 1) {
            newrow= 3;
        }
        else if(newrow == 3) {
            newrow= 0;
        }
        else {
            newrow++;
        }
        break;

    // move highlight bar up

    case UP_ARROW:
    case UP_ARROW_K:
        if(newrow == 0) {
            newrow= 3;
        }

```

7-2 Continued.

```
        }
        else if(newrow == 3) {
            newrow= 1;
        }
        else {
            newrow--;
        }
        break;
    }

    } while(!exit_flag);

    if(mouse_installed) {
        ms_off();
    }

    restRect(R);
    dsyRect(R);

//
// return menu bar to original state
//
    print_menu_bar();

    if(mouse_installed) {
        ms_on();
    }

//
// return the key press
//

    return key;
}

//
////////////////////////////////////
////////////////////////////////////
//
// open Print drop down window
//

int openDD_Print(int num_copies)
{
    UCHAR attr1, attr2, exit_flag= 0;
    RECT *R;
    char buffer[20];
    int key, oldrow= 0, newrow= 0;
    int x, y;

//
// set attributes
//

    attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);
    attr2= mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK);
```

```

//
// initialize the rectangular structure
//

R= setRect(R,
           1,
           6,
           1 + 3,
           6 + 22);

//
// save the screen image under the rectangle
//

saveRect(R);

//
// draw a rectangular box under the File option
// using a Single sided border (S_S_S_S);
//

boxRect(R,
        S_S_S_S,
        attr1);

//
// attributes change on Print option
//

scrn_attr(0, 6, 7, attr1);

// write the menu items to the screen

scrn_write(2,
           1 + 6,
           0,
           " Number of Copies   ",
           attr1);

memset(buffer, 0, 10);
sprintf(buffer, "% 3d", num_copies);
scrn_write(2,
           1 + 6 + 18,
           0,
           buffer,
           attr1);

scrn_write(3,
           1 + 6,
           0,
           " Start Print Job... ",
           attr1);

scrn_attr(oldrow + 2, 2 + 6, 21, attr1);

scrn_attr(2,
           2 + 6,
           1,
           attr2);

```

7-2 Continued.

```
    scrn_attr(3,
              2 + 6,
              1,
              attr2);

// draw highlight bar

    scrn_attr(newrow + 2, 1 + 6, 21, mk_attr_inverse(attr1));

    if(mouse_installed) {
        ms_on();
    }

//
//  main keyboard loop
//

    do {
        if(oldrow != newrow) {

            if(mouse_installed) {
                ms_off();
            }

            // write the menu items to the screen

            scrn_write(2,
                      1 + 6,
                      0,
                      " Number of Copies    ",
                      attr1);

            memset(buffer, 0, 10);
            sprintf(buffer, "% 3d", num_copies);
            scrn_write(2,
                      1 + 6 + 18,
                      0,
                      buffer,
                      attr1);

            scrn_write(3,
                      1 + 6,
                      0,
                      " Start Print Job... ",
                      attr1);

            scrn_attr(oldrow + 2, 2 + 6, 21, attr1);

            scrn_attr(2,
                      2 + 6,
                      1,
                      attr2);

            scrn_attr(3,
                      2 + 6,
                      1,
                      attr2);
```

```

        // draw highlight bar

        scrn_attr(newrow + 2, 1 + 6, 21, mk_attr_inverse(attr1));

        oldrow= newrow;

        if(mouse_installed) {
            ms_on();
        }
    }

    key= kb_status();

    if((!key) && (mouse_installed)) {

        key= ms_status(&x, &y);

        if((key == 1) && (x >= 56) && (x <= (56 + (21 * 7)))
            && (y == 16)) {
            key= ALT_N;
        }

        if((key == 1) && (x >= 56) && (x <= (56 + (21 * 7)))
            && (y == 24)) {
            key= ALT_S;
        }

        if((key == 1) && (y >= 8)) {
            key= ESCAPE;
        }

        if((key == 2) && (y >= 8)) {
            key= RIGHT_ARROW;
        }

    }

    // process key press

    switch(key) {
        // Item selected so
        // break from the loop

        case ENTER:
            if(newrow == 1) {
                key= ALT_X;
            }
            else {
                key= ALT_0;
            }
            exit_flag= 1;
            break;

        // select file for printing
        // quit program
        // no action

        case RIGHT_ARROW:
        case RIGHT_ARROW_K:
    
```

7-2 Continued.

```
        case LEFT_ARROW:
        case LEFT_ARROW_K:
        case ALT_N:
        case ALT_S:
        case ESCAPE:
            exit_flag= 1;
            break;

        // move highlight bar down

        case DOWN_ARROW:
        case DOWN_ARROW_K:
            if(newrow == 1) {
                newrow= 0;
            }
            else {
                newrow++;
            }
            break;

        // move highlight bar up

        case UP_ARROW:
        case UP_ARROW_K:
            if(newrow == 0) {
                newrow= 1;
            }
            else {
                newrow--;
            }
            break;
    }

    } while(!exit_flag);

//
// if the mouse is installed turn the mouse off
//

    if(mouse_installed) {
        ms_off();
    }

    restRect(R);

    dsyRect(R);

//
// return menu bar to original state
//
    print_menu_bar();

    if(mouse_installed) {
        ms_on();
    }

//
// return the key press
//
```



```

        return key;
    }

    //
    ////////////////////////////////////////////////////
    ////////////////////////////////////////////////////
    //
    // open Options drop down window
    //

    int openDD_Options()
    {
        UCHAR attr1, attr2, exit_flag= 0;
        RECT  *R;
        int    key, oldrow= 0, newrow= 0;
        int    x, y;

        //
        // set attributes
        //

        attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);
        attr2= mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK);

        //
        // initialize the rectangular structure
        //

        R= setRect(R,
                   1,
                   13,
                   1 + 4,
                   13 + 24);

        //
        // save the screen image under the rectangle
        //

        saveRect(R);

        //
        // draw a rectangular box under the File option
        // using a Single sided border (S_S_S_S);
        //

        boxRect(R,
                S_S_S_S,
                attr1);

        //
        // attributes change on Options option
        //

        scrn_attr(0, 13, 9, attr1);

        // write the menu items to the screen

        scrn_write(2,

```

7-2 Continued.

```
        1 + 13,
        0,
        " Select Printer  LPT1  ",
        attr1);

scrn_write(3,
        1 + 13,
        0,
        " Select Print Options  ",
        attr1);

scrn_write(4,
        1 + 13,
        0,
        " Install Printer Codes ",
        attr1);

scrn_attr(oldrow + 2, 2 + 13, 23, attr1);

scrn_attr(2,
        2 + 13 + 7,
        1,
        attr2);

scrn_attr(3,
        2 + 13,
        1,
        attr2);

scrn_attr(4,
        2 + 13,
        1,
        attr2);

// draw highlight bar

scrn_attr(newrow + 2, 1 + 13, 23, mk_attr_inverse(attr1));

if(mouse_installed) {
    ms_on();
}

//
//  main keyboard loop
//

do {

    if(oldrow != newrow) {

        if(mouse_installed) {
            ms_off();
        }

        // write the menu items to the screen

        scrn_write(2,
            1 + 13,
```

```

        0,
        " Select Printer  LPT1  ",
        attr1);

scrn_write(3,
           1 + 13,
           0,
           " Select Print Options  ",
           attr1);

scrn_write(4,
           1 + 13,
           0,
           " Install Printer Codes  ",
           attr1);


scrn_attr(oldrow + 2, 2 + 13, 23, attr1);

scrn_attr(2,
          2 + 13 + 7,
          1,
          attr2);

scrn_attr(3,
          2 + 13,
          1,
          attr2);

scrn_attr(4,
          2 + 13,
          1,
          attr2);

// draw highlight bar

oldrow= newrow;

scrn_attr(newrow + 2, 1 + 13, 23, mk_attr_inverse(attr1));

if(mouse_installed) {
    ms_on();
}

}

key= kb_status();

if((!key) && (mouse_installed)) {

    key= ms_status(&x, &y);

    if((key == 1) && (x >= 112) && (x <= (112 + (24 * 8)))
        && (y == 16)) {
        key= ALT_P;
    }

    if((key == 1) && (x >= 112) && (x <= (112 + (24 * 8)))
        && (y == 24)) {
        key= ALT_S;
    }
}

```

7-2 Continued.

```
        if((key == 1) && (x >= 112) && (x <= (112 + (21 * 7)))
            && (y == 32)) {
            key= ALT_I;
        }

        if((key == 1) && (y >= 8)) {
            key= ESCAPE;
        }

        if((key == 2) && (y >= 8)) {
            key= RIGHT_ARROW;
        }
    }

    // process key press
    switch(key) {
        // Item selected so
        // break from the loop

        case ENTER:
            if(newrow == 1) {
                key= ALT_S;
            }
            else if(newrow == 2) {
                key= ALT_I;
            }
            else {
                key= ALT_P;
            }
            exit_flag= 1;
            break;

        // select file for printing
        // quit program
        // no action

        case RIGHT_ARROW:
        case RIGHT_ARROW_K:
        case LEFT_ARROW:
        case LEFT_ARROW_K:
        case ALT_P:
        case ALT_S:
        case ALT_I:
        case ESCAPE:
            exit_flag= 1;
            break;

        // move highlight bar down

        case DOWN_ARROW:
        case DOWN_ARROW_K:
            if(newrow == 2) {
                newrow= 0;
            }
            else {
                newrow++;
            }
            break;
```

```

        // move highlight bar up

        case UP_ARROW:
        case UP_ARROW_K:
            if(newrow == 0) {
                newrow= 2;
            }
            else {
                newrow--;
            }
            break;
    }

    } while(!exit_flag);

//
// if the mouse is installed turn the mouse off
//

    if(mouse_installed) {
        ms_off();
    }

    restRect(R);

    dsyRect(R);

//
// return menu bar to original state
//
    print_menu_bar();

    if(mouse_installed) {
        ms_on();
    }

//
// return the key press
//

    return key;
}

//
////////////////////////////////////
////////////////////////////////////
//
// open help drop down window
//

int openDD_Help()
{
    UCHAR attr1, attr2, exit_flag= 0;
    RECT *R;
    int key, oldrow= 0, newrow= 0;
    int x, y;

//
// set attributes

```

7-2 Continued.

```
//  
  
    attr1= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);  
    attr2= mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK);  
  
//  
// initialize the rectangular structure  
//  
  
    R= setRect(R,  
              1,  
              22,  
              1 + 3,  
              22 + 19);  
  
//  
// save the screen image under the rectangle  
//  
  
    saveRect(R);  
  
//  
// draw a rectangular box under the File option  
// using a Single sided border (S_S_S_S);  
//  
  
    boxRect(R,  
            S_S_S_S,  
            attr1);  
  
//  
// attributes change on Help option  
//  
  
    scrn_attr(0, 22, 6, attr1);  
  
// write the menu items to the screen  
  
    scrn_write(2,  
              1 + 22,  
              0,  
              " Program Help... ",  
              attr1);  
  
    scrn_write(3,  
              1 + 22,  
              0,  
              " Dranoel Help... ",  
              attr1);  
  
  
    scrn_attr(oldrow + 2, 2 + 22, 5, attr1);  
  
    scrn_attr(2,  
              2 + 22,  
              1,  
              attr2);  
  
    scrn_attr(3,  
              2 + 22,
```

```

        1,
        attr2);

// draw highlight bar

scrn_attr(newrow + 2, 1 + 22, 18, mk_attr_inverse(attr1));

if(mouse_installed) {
    ms_on();
}
//
// main keyboard loop
//

do {

    if(oldrow != newrow) {

        if(mouse_installed) {
            ms_off();
        }

        // write the menu items to the screen

        scrn_write(2,
                    1 + 22,
                    0,
                    " Program Info... ",
                    attr1);

        scrn_write(3,
                    1 + 22,
                    0,
                    " Dranoel Info... ",
                    attr1);

        scrn_attr(oldrow + 2, 2 + 22, 5, attr1);

        scrn_attr(2,
                    2 + 22,
                    1,
                    attr2);

        scrn_attr(3,
                    2 + 22,
                    1,
                    attr2);

        // draw highlight bar

        scrn_attr(newrow + 2, 1 + 22, 18, mk_attr_inverse(attr1));

        oldrow= newrow;

        if(mouse_installed) {
            ms_on();
        }
    }
}

```

7-2 Continued.

```
key= kb_status();

if((!key) && (mouse_installed)) {

    key= ms_status(&x, &y);

    if((key == 1) && (x >= 184) && (x <= 328)
        && (y == 16)) {
        key= ALT_P;
    }

    if((key == 1) && (x >= 184) && (x <= 328)
        && (y == 24)) {
        key= ALT_M;
    }

    if((key == 1) && (y >= 8)) {
        key= ESCAPE;
    }

    if((key == 2) && (y >= 8)) {
        key= RIGHT_ARROW;
    }

}

// process key press

switch(key) {
    // Item selected so
    // break from the loop

    case ENTER:
        if(newrow == 1) {
            key= ALT_P;
        }
        else {
            key= ALT_M;
        }
        exit_flag= 1;
        break;

    // select program help
    // select mentaur help
    // no action

    case RIGHT_ARROW:
    case RIGHT_ARROW_K:
    case LEFT_ARROW:
    case LEFT_ARROW_K:
    case ALT_P:
    case ALT_M:
    case ESCAPE:
        exit_flag= 1;
        break;

    // quit program

    case ALT_X:
```



```

        exit_flag= 1;
        break;

    // move highlight bar down

    case DOWN_ARROW_K:
    case DOWN_ARROW:
        if(newrow == 1) {
            newrow= 0;
        }
        else {
            newrow++;
        }
        break;

    // move highlight bar up

    case UP_ARROW:
    case UP_ARROW_K:
        if(newrow == 0) {
            newrow= 1;
        }
        else {
            newrow--;
        }
        break;
    }
} while(!exit_flag);

//
// if the mouse is installed then turn it off
//

    if(mouse_installed) {
        ms_off();
    }

    restRect(R);

    dsyRect(R);

//
// return menu bar to original state
//
    print_menu_bar();

    if(mouse_installed) {
        ms_on();
    }

//
// return the key press
//

    return key;

}

//
////////////////////////////////////

```

7-2 Continued.

```
////////////////////////////////////////
//
// main function
//

void main()
{
    UCHAR    attr2, attr3, attr4;
    UCHAR    exit_flag= 0, list_created= 0;
    WIND      *Wptr, *Wptr2;
    FILE      *fptr;
    ULONG     file_length;
    ULONG     chars_printed= 0;
    struct    PrintOpt P01;
    char      **cptrptr, *ch, *blk;
    char      *file_buffer;
    char      buffer[60];
    int       key, ret_val, num_print_copies= 1;
    int       x, y, counter, counter1;
    int       count, entries, ctr;
    int       LF_counter= 0;
    int       page_length= 58;
    char      ascii_buffer[20];
    int       page_number= 1;

    //
    // initialize print options members
    //

    P01.compress=      0;
    P01.double_strike= 0;
    P01.header=        0;
    P01.left_offset=   0;

    //
    // initialize the screen
    //
    scrn_init();

    //
    // initialize the mouse
    //

    mouse_installed= ms_init();

    //
    // initialize attributes
    //

    attr2= mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK);
    attr3= mk_attr(RED, WHITE, OFF_INTENSITY, ON_BLINK);

    //
    // save the current cursor location
    //

    cu_save_loc();

    //
    // save the current cursor size
```

```

//
    cu_save_size();

//
// save the screen
//

    scrn_save();

//
// turn off the cursor and mouse
//

    cu_remove();
    if(mouse_installed) {
        ms_off();
    }

//
// clear the screen
//

    scrn_clear();

//
// alter screen attributes
//

    scrn_change_attr(attr2);

//
// print menu bar
//

    print_menu_bar();

//
// fill in block characters
//
    for(counter= 1; counter < 24; counter++) {
        scrn_repeat_char(counter,
                        0,
                        80,
                        177,
                        mk_attr(BLACK, WHITE, OFF_INTENSITY, OFF_BLINK));
    }

//
// print exit message at the screen bottom
// lower right
//

    scrn_write(24,
                58,
                0,
                "Leave Program: ALT-X",

```

7-2 Continued.

```
        mk_attr(RED, WHITE, OFF_INTENSITY, OFF_BLINK));

//
// turn on the mouse if installed
//

    if(mouse_installed) {
        ms_on();
    }

//
// main program loop
//

do {
    // get the keyboard - no wait for key press

    key= kb_status();

    // if there is no key press and the
    // mouse is installed then

    if((!key) && (mouse_installed)) {

        // check the status of the mouse

        key= ms_status( &x, &y);

        // if there is a left button press and
        // the mouse is at the top of the screen
        // then (using the data that you obtained
        // from PROG5-2) map the mouse location
        // to the appropriate key press

        if((key == 1) && (y == 0)) {

            if((x >= 0) && (x <= 40)) {
                key= ALT_F;
            }

            if((x >= 48) && (x <= 96)) {
                key= ALT_P;
            }

            if((x >= 104) && (x <= 168)) {
                key= ALT_0;
            }

            if((x >= 176) && (x <= 216)) {
                key= ALT_H;
            }
        }

        //////////////////////////////////////
        // check to see if ALX-X

        if((key == 1) && (y == 192) && (x >= 456) && (x <= 632)) {
            key= ALT_X;
        }
    }
}
```

```

    }

// process key press

new_window:

switch(key) {

    case ALT_X:
        exit_flag= 1;
        break;

    case ALT_F:
        if(mouse_installed) {
            ms_off();
        }

        scrn_write(24, 0, 0, ERASE_message, attr2);

        ret_val= openDD_File();

        if((ret_val == RIGHT_ARROW) || (ret_val == RIGHT_ARROW_K)) {
            key = ALT_P;
            goto new_window;
        }

        else if((ret_val == LEFT_ARROW) || (ret_val == LEFT_ARROW_K)) {
            key = ALT_H;
            goto new_window;
        }

        else if(ret_val == ALT_O) {
            cptrptr= FILE_Open_ALT_O(&ret_val, &entries);
            if(ret_val == K_N) {
                list_created= 1;
                if(mouse_installed) {
                    ms_off();
                }
                Wptr= FILE_display_files((char **)cptrptr, (int) entries);
                if(mouse_installed) {
                    ms_on();
                }
            }
            else {
                for(count= 0; count < entries; count++) {
                    if(*(cptrptr + (count * sizeof(char *))) != 0) {
                        free((void *)*(cptrptr + (count * sizeof(char *))));
                    }
                }
                free((void *)cptrptr);
            }
        }

        else if(ret_val == ALT_X) {
            scrn_write(24, 0, 0, FILE_ALT_X, attr2);
        }

```

7-2 Continued.

```
        else {
            FILE_About_ALT_A();
        }

        if(mouse_installed) {
            ms_on();
        }

        if(ret_val == ALT_X) {
            exit_flag= 1;
        }
        break;
case ALT_P:
    if(mouse_installed) {
        ms_off();
    }

    scrn_write(24, 0, 0, ERASE_message, attr2);

    ret_val= openDD_Print(num_print_copies);

    if((ret_val == RIGHT_ARROW) || (ret_val == RIGHT_ARROW_K)) {
        key = ALT_O;
        goto new_window;
    }

    else if((ret_val == LEFT_ARROW) || (ret_val == LEFT_ARROW_K)) {
        key = ALT_F;
        goto new_window;
    }

    else if(ret_val == ALT_N) {
        scrn_write(24, 0, 0, PRINT_ALT_N, attr2);
    }

    else {

        // open print window here

        if(list_created) {

            Wptr2= PRINT_Start_ALT_S();

            // allocate memory for file_buffer

            file_buffer= (char *)malloc(FILEMAX);
            ch= file_buffer;

            for(counter1= 0; counter1 < entries; counter1++) {

                // file print here

                wind_write(Wptr2, 2, 12, 0, *(cptrptr + (counter1 *
                    sizeof(char *))), attr2);

                // initialize file buffer
```

```

memset(file_buffer, 0, FILEMAX);

// open the file for read
fptr= fopen(*(cptrptr + (counter1 * sizeof(char *))),
            "r");

// read the file into the file buffer
fread(file_buffer, FILEMAX, 1, fptr);

// determine the length of the file
file_length= strlen(file_buffer);

wind_write(Wptr2, 3, 13, 0, (char *)itoa(file_length,
            buffer, 10), attr2);

wind_write(Wptr, 4 + counter1, 26, 0, "Printing...",
            attr3);

////////////////////////////////////

print_open(0);

if(P01.compress) {
    print_char(0, PRINTER_CONDENSE_ON);
}

if(P01.double_strike) {
    print_char(0, aESC);
    print_char(0, PRINTER_DOUBLE_STRIKE_ON);
}

if(P01.header) {
    print_char(0, aCR);
    print_string(0, "File Name: ");
    print_string(0,
                *(cptrptr +
                (counter1 * sizeof(char *))));
    if(P01.compress) {
        print_set_column(0, 116);
    }
    else {
        print_set_column(0, 60);
    }
    print_string(0, "Page Number: ");
    itoa(page_number, ascii_buffer, 10);
    page_number++;
    print_string(0, ascii_buffer);
    print_char(0, aLF);
    print_set_column(0, 0);
    if(P01.compress) {
        for(ctr= 0; ctr < 136; ctr++) {
            print_char(0, '_');
        }
    }
    else {
        for(ctr= 0; ctr < 80; ctr++) {
            print_char(0, '_');
        }
    }
}

```

7-2 Continued.

```
    }
    print_char(0, aLF);
    print_char(0, aLF);
    }

    //////////////////////////////////////
    //
    // main printer loop
    //

    for(counter= 0; counter < file_length; counter ++ ) {

        //////////////////////////////////////
        //
        // Routine to write file name and
        // page number as header of each
        // page
        //

        if(*ch == aLF) {
            if(LF_counter >= page_length) {
                LF_counter= 0;
                print_char(0, PRINTER_FORM_FEED);
                if(P01.header) {
                    print_char(0, aCR);
                    print_string(0,"File Name: ");
                    print_string(0,
                                *(cptrptr +
                                   (counter1 *
                                    sizeof(char *)))));
                    if(P01.compress) {
                        print_set_column(0, 116);
                    }
                    else {
                        print_set_column(0, 60);
                    }
                    print_string(0, "Page Number: ");
                    itoa(page_number, ascii_buffer, 10);
                    page_number++;
                    print_string(0, ascii_buffer);
                    print_char(0, aLF);
                    print_set_column(0, 0);
                    if(P01.compress) {
                        for(ctr= 0; ctr < 136; ctr++) {
                            print_char(0, '_');
                        }
                    }
                    else {
                        for(ctr= 0; ctr < 80; ctr++) {
                            print_char(0, '_');
                        }
                    }
                    print_char(0, aLF);
                    print_char(0, aLF);
                }
            }
            else {
                LF_counter++;
            }
        }
    }
```



```

    }
    print_char(0, *ch++);
    chars_printed++;
    wind_write(Wptr2,
               4,
               27,
               0,
               (char *)ittoa(chars_printed,
                             buffer,
                             10),
               attr2);
    }

    if(P01.double_strike) {
        print_char(0, aESC);
        print_char(0, PRINTER_DOUBLE_STRIKE_OFF);
    }

    if(P01.compress) {
        print_char(0, PRINTER_CONDENSE_OFF);
    }

    print_char(0, PRINTER_FORM_FEED);
    print_close(0);

    wind_write(Wptr, 4 + counter1, 26, 0, "All Done  ",
               attr2);

    // close the file
    fclose(fp);
}

// free the buffer
free(file_buffer);

wind_write(Wptr2, 5, 2, 0, "Print Job Completed", attr2);
kb_read();

wind_remove(Wptr);
wind_destroy(Wptr);

wind_remove(Wptr2);
wind_destroy(Wptr2);

list_created= 0;
}

}

if(mouse_installed) {
    ms_on();
}
break;

```

7-2 Continued.

```
case ALT_0:
    if(mouse_installed) {
        ms_off();
    }

    scrn_write(24, 0, 0, ERASE_message, attr2);

    ret_val= openDD_Options();

    if((ret_val == RIGHT_ARROW) || (ret_val == RIGHT_ARROW_K)) {
        key = ALT_H;
        goto new_window;
    }

    else if((ret_val == LEFT_ARROW) || (ret_val == LEFT_ARROW_K)) {
        key = ALT_P;
        goto new_window;
    }

    else if(ret_val == ALT_P) {
        scrn_write(24, 0, 0, OPTIONS_ALT_P, attr2);
    }

    else if(ret_val == ALT_S) {
        key= OPTIONS_Set_ALT_S(&P01);
        // action here
    }

    else {
        scrn_write(24, 0, 0, OPTIONS_ALT_I, attr2);
    }

    if(mouse_installed) {
        ms_on();
    }
    break;

case ALT_H:
    if(mouse_installed) {
        ms_off();
    }

    scrn_write(24, 0, 0, ERASE_message, attr2);

    ret_val= openDD_Help();

    if((ret_val == RIGHT_ARROW) || (ret_val == RIGHT_ARROW_K)) {
        key = ALT_F;
        goto new_window;
    }

    else if((ret_val == LEFT_ARROW) || (ret_val == LEFT_ARROW_K)) {
        key = ALT_0;
        goto new_window;
    }

    else if(ret_val == ALT_P) {
        scrn_write(24, 0, 0, HELP_ALT_P, attr2);
    }
```

```

        }

        else {
            scrn_write(24, 0, 0, HELP_ALT_M, attr2);
        }

        if(mouse_installed) {
            ms_on();
        }
        break;
    }

    } while(!exit_flag);

//
// turn off the mouse
//

    if(mouse_installed) {
        ms_off();
    }

//
// restore the screen
//

    scrn_restore();

//
// restore cursor location
//

    cu_rest_loc();

//
// restore previously saved cursor size
//

    cu_rest_size();

//
// display the cursor
//

    cu_display();

}

//
////////////////////

```

Summary

This chapter presented many useful printer control management functions. These functions allow you to:

- Send a character to the printer
- Send a string of characters to the printer
- Set the print head to a specified column location

Chapter 7 completes the presentation of the demonstration programs for the OS/2 and DOS C libraries presented in this book. Part 2 presents the source code for the OS/2 version of the functions presented in the library. Part 3 presents the source code for the DOS version library.

Part Two

OS/2 full screen character mode library

Eight source code modules make up the OS/2 Library. Keyboard, mouse, and video access is handled via OS/2's VIO, KBD, and MOUS calls. Marc Neuberger and I have assembled the full documentation to the VIO, KBD, and MOUS function calls, and the information will be presented (with IBM's blessing) in a forthcoming Windcrest book.

8

OS/2 full screen keyboard management functions

This chapter presents the source code to the OS/2 keyboard management object module that has been placed in the OS2TEXT.LIB library file. Table 8-1 repeats the keyboard function prototypes.

Table 8-1 Keyboard function prototypes.

int	kb_edit(char *response, int row, int column, int dlen, int opt, UCHAR attr);
int	kb_read(void);
int	kb_status(void);
char	kb_char(void);
UCHAR	kb_scan(void);

Figure 8-1 presents the OS/2 based source code listing to KEYBOARD.C. This source file uses OS/2's KBD functions as building blocks for the migration library's higher level keyboard routines.

8-1 The source code listing to KEYBOARD.C.

```
////////////////////////////////////
//
// keyboard.c
//
// OS/2 Version
//
////////////////////////////////////

////////////////////////////////////
//
// OS/2 defines here
//

#define INCL_VIO
#define INCL_KBD
#define INCL_MOU
#define INCL_DOSPROCESS
#define INCL_DOSSEMAPHORES

////////////////////////////////////
//
// includes here
//

#include <os2.h>
#include <string.h>
#include <ctype.h>

#include "tproto.h"

////////////////////////////////////
//
// kb_status
//
// Checks for a key press and returns
// in a 16-bit format an 8-bit char
// and 8-bit unsigned char scan codes
// (Does not stop program execution..
// facilitates polling)
//

int kb_status()
{
    KBDKEYINFO key_code;

    key_code.chScan= 0;
    key_code.chChar= 0;

    KbdCharIn(&key_code, IO_NOWAIT, 0);

    return(mk_token(key_code.chScan, key_code.chChar));
}

////////////////////////////////////
//
// kb_read
//
// Checks for a key press and returns
```



```

// in a 16-bit format an 8-bit char
// and 8-bit unsigned char scan codes
// (Stop program execution.._
//

int kb_read()
{
    KBDKEYINFO key_code;

    KbdCharIn(&key_code, 0, 0);
    return(mk_token(key_code.chScan, key_code.chChar));
}

////////////////////////////////////
//
// kb_char
//
// Checks for a key press and returns
// an 8-bit char
//

UCHAR kb_char()
{
    KBDKEYINFO key_code;

    KbdCharIn(&key_code, 0, 0);

    return(key_code.chChar);
}

////////////////////////////////////
//
// kb_scan
//
// Checks for a key press and returns
// an 8-bit scan code
//

UCHAR kb_scan()
{
    KBDKEYINFO key_code;

    KbdCharIn(&key_code, 0, 0);
    return(key_code.chScan);
}

////////////////////////////////////
//
// kb_edit
//
// Alpha-numeric field entry routine
//

int kb_edit(char *response,
            int row,
            int column,
            int dlen,
            int opt,
            UCHAR attr)

```

8-1 Continued.

```
{
int key;
int start, stop;
char *rptr;
int i;
int ins=0;
char buf[80];
int cur, start_column;
int ret_val;
int tlen;

    // set start column for stopper on left arrow
    start_column= column;

    // save cursor shape and location
    cur= cu_get_shape();
    cu_save_loc();

    // turn the cursor on
    cu_display();

    // place '\0' at end point of response buffer
    *(response + dlen)= 0;

    // make response string upper or lower
    switch(opt)
    {
        case LOWER:
            strlwr(response);
            break;
        case UPPER:
            strupr(response);
            break;
        case NAME:
            response= strlwr(response);
            *response= toupper(*response);
            break;
    }

    // copy contents of response buffer to buf[]
    for(i= 0; i < dlen; i++) {
        buf[i]= response[i];
    }

    // set start and stop variables
    start= column;
    stop= start + dlen;

    // alter screen attributes for edit field
    scrn_attr(row, column, dlen, attr);
```

```

// if *response != 0 then write string to screen
if(*response) {
    scrn_write(row, column, 0, response, attr);
}

// adjust cursor location to response string end
cu_move(row, column+= strlen(response));

// set response pointer to end of response buffer
rptr= response + strlen(response);

// wait for key press
key= kb_read();

// process key press first time through
switch(key)
{
    case ESCAPE:
    case ENTER:
    case F1:
    case F2:
    case F3:
    case F4:
    case F5:
    case F6:
    case F7:
    case F8:
    case F9:
    case F10:
    case UP_ARROW:
    case DOWN_ARROW:
    case UP_ARROW_K:
    case DOWN_ARROW_K:
    case TAB:
        cu_set_shape(cur); // restore cursor shape
        cu_rest_loc(); // restore cursor location
        return(key); // return key press value

    case HOME:
    case PGUP:
        memset(response, 0, dlen + 1); // clear response buffer
        rptr= response; // reset pointer
        scrn_write(row, start, dlen, response, attr); // clr scrn
        column= start; // reset column to start
        cu_move(row, column); // adjust cursor location
        break;

    case CNTL_LEFTA:
        while(*--rptr!=' '); // mv ptr to ' ' char
        rptr++; // and incr ptr by 1

        // adjust cursor if ' ' is after start

        if(start_column<start+(int)(rptr-response)) {
            cu_move(row, column= start + (int)(rptr - response));
        }
}

```

8-1 Continued.

```
        else {
            // set column & adjust cursor

            column= start_column;
            cu_move(row, start_column);
        }

        // write response buffer to screen

        scrn_write(row, start, dlen, response, attr);
        break;

    case LEFT_ARROW:
    case LEFT_ARROW_K:
        // is cursor right of start?

        if(start_column < column) {
            // yes -> adjust cursor left

            cu_move(row, --column);

            // adjust ptr

            rptr--;
        }

    case END:
    case INSERT:
    case DELETE:
    case RIGHT_ARROW:
    case RIGHT_ARROW_K:
    case PGDN:

        // write response buffer to screen

        scrn_write(row, start, dlen, response, attr);
        break;

    default:

        // NULL out scan code

        key&= 0x00ff;

        // set letter case

        switch(opt)
        {
            case LOWER:
                key= tolower(key);
                break;

            case UPPER:
                key= toupper(key);
                break;
        }

        // if printable character

        if((key >= 0x20) && (key <= 0x7e)) {
```

```

        // NULL buffer
        memset(response, 0, dlen + 1);
        // set pointer to response start
        rptr= response;
        // place key in response
        *rptr++= (char)key;
        // write buffer to screen
        scrn_write(row, start, dlen, response, attr);
        // adjust column pointer
        column= start + 1;
        // adjust cursor position
        cu_move(row, column);
    }

    // is the key a backspace?
    if(key == aBS) {
        // if column is greater than start
        if(column > start) {
            // backspace in response buffer

            rptr--;

            // place NULL at new location
            *rptr= 0;

            // write response buffer
            scrn_write(row, start, dlen, response, attr);

            // adjust cursor location
            cu_move(row, --column);
        }
        break;
    }
}

// process key press from now on
do
{
    // adjust case and write
    if(opt == NAME) {
        *response= toupper(*response);
        scrn_write(row, start, dlen, response, attr);
    }
}

```

8-1 Continued.

```
    }

    // stop and wait for key press

    key= kb_read();

    // process key press

    switch(key)
    {
        case F1:                // return from vedit
        case F2:                // and report key press
        case F3:
        case F4:
        case F5:
        case F6:
        case F7:
        case F8:
        case F9:
        case F10:
        case UP_ARROW:
        case DOWN_ARROW:
        case UP_ARROW_K:
        case DOWN_ARROW_K:
        case TAB:
        case ENTER:
            cu_set_shape(cur);
            cu_rest_loc();
            return key;

        case ESCAPE:
            cu_set_shape(cur);
            cu_rest_loc();
            for(i= 0; i < dlen; i++) { // restore original
                response[i]= buf[i]; // buffer contents
            }
            return key;

        case CNTL_G:
        case DELETE:            // delete char and adjust buffer
            for(i= 0; i < stop - column + 1; i++) {
                *(rptr + i)= *(rptr + 1 + i);
            }
            *(rptr - 1 + i)= 0;
            scrn_write(row, start, dlen, response, attr);
            break;

        case CNTL_T:            // erase from cursor to
            while(*rptr && *rptr != ' ') // EOL
                for(i= 0; i < stop - column + 1; i++)
                    *(rptr + i)= *(rptr + 1 + i);
            while(*rptr && *rptr == ' ')
                for(i= 0; i < stop - column + 1; i++)
                    *(rptr + i)= *(rptr + 1 + i);
            scrn_write(row, start, dlen, response, attr);
            break;

        case CNTL_END:          // erase from cursor to entry end
            memset(rptr, 0, stop - column);
            scrn_write(row, start, dlen, response, attr);
    }
```

```

        break;

case LEFT_ARROW:    // move cursor left
case LEFT_ARROW_K:
    if(start_column < column) {
        cu_move(row, --column);
        rptr--;
    }
    break;

case CNTL_LEFTA:    // move by word left
    if(rptr == response) {
        break;
    }
    while(*--rptr == ' ' && (int)(rptr - response) > 0);
    while(*--rptr != ' ' && (int)(rptr - response) > 0);
    if((int)(rptr - response) > 0) {
        rptr++;
    }
    cu_move(row, column= start + (int)(rptr - response));
    break;

case CNTL_RIGHTA:   // move by word right
    if(*rptr) {
        while (*++rptr != ' ' && *rptr);
    }
    if(*rptr) {
        while (*++rptr == ' ' && *rptr);
    }
    cu_move(row, column= start + (int)(rptr - response));
    break;

case RIGHT_ARROW:   // move cursor right
case RIGHT_ARROW_K:
    if (*rptr) {
        cu_move(row, ++column);
        rptr++;
    }
    break;

case CNTL_BS:       // erase entry and start over
    memset(response, 0, dlen + 1);
    rptr= response;
    scrn_write(row, start, dlen, response, attr);
    column= start;
    cu_move(row,column);
    break;

case HOME:          // go to beginning of entry
    cu_move(row ,column= start);
    rptr= response;
    break;

case END:           // go to end of entry
    cu_move(row, column= start + strlen(response));
    rptr= response + strlen(response);
    break;

case CNTL_H:
case BS:            // move cursor back and delete
    if(column>start) {

```

8-1 Continued.

```
        rptr--;
        for (i= 0; i < stop - column + 1; i++) {
            *(rptr + i)= *(rptr + 1 + i);
        }
        scrn_write(row, start, dlen, response, attr);
        cu_move(row, --column);
    }
    else {
        bleep();
    }
    break;

case INSERT:          // toggle insert and overlay mode
    if(ins)           // cursor size adjusted
    {
        ins=0;
        cu_rest_size();
    }
    else {
        ins= 1;
        cu_save_size();
        cu_set_size(0, 14);
    }
    break;
default:

    // mask 8 bits

    key&= 0x00ff;

    // process option

    switch (opt)
    {
        case NAME:
        case LOWER:

            // force lower case

            key= tolower(key);
            break;

        case UPPER:

            // force upper case

            key= toupper(key);
            break;

    }

    key&= 0x00ff;

    // is key printable character?

    if((key >= 0x20) && (key <= 0x7d)) {
        // is insert key toggled on

        if(ins) {
            // determine length of response
```



```

        tlen= strlen(response);
        // is response less than max response?
        if(tlen < dlen) {
            // relocate string making
            // space for new key

            for (i= dlen; i > (rptr-response); i--)
                response[i]= response[i - 1];

            // write response buffer

            scrn_write(row, start, dlen, response, attr);
        }
    }

    // is end of edit field not reached?
    if(column < stop) {
        // write char to screen

        scrn_char(row, column, key, attr);

        // place key in buffer
        *rptr++= (char)key;

        // adjust column 1 right
        column++;

        // move the cursor on the screen
        cu_move(row, column);
    }
    else
        // bleep at field end
        bleep();
}
} while(1);
}

```

8-1 Ends.

9

OS/2 full screen cursor management functions

This chapter presents the source code to the OS/2 cursor management object module that has been placed in the OS2TEXT.LIB library file. Table 9-1 repeats the cursor function prototypes.

**Table 9-1 Cursor
function prototypes.**

void	cu_get_loc(int *, int *);
int	cu_get_shape(void);
void	cu_move(int, int);
void	cu_relative_move(int, int);
void	cu_remove(void);
void	cu_display(void);
void	cu_save_size(void);
void	cu_rest_size(void);
void	cu_set_size(int, int);
void	cu_save_loc(void);
void	cu_set_shape(int);
void	cu_rest_loc(void);

Figure 9-1 presents the OS/2 based source code listing to CURSOR.C. This source file uses OS/2's VIO functions as building blocks for the migration library's higher level cursor management routines.

9-1 The source code listing to CURSOR.C.

```
////////////////////////////////////////
//
// cursor.c
//
////////////////////////////////////////

////////////////////////////////////////
//
// defines required for OS/2
//

#define INCL_VIO
#define INCL_KBD
#define INCL_MOU
#define INCL_DOSPROCESS
#define INCL_DOSSEMAPHORES

////////////////////////////////////////
//
// include files
//

#include <os2.h>
#include "tproto.h"

////////////////////////////////////////
//
// global variables
//

USHORT      c_row, c_col;
VIOC_CURSORINFO save_restore_state;
VIOC_CURSORINFO ssize_sr;
VIOC_CURSORINFO off_and_on;
VIOC_CURSORINFO off;
VIOC_CURSORINFO vioc_temp;
int         G_CURSOR_VISIBLE= 1;

////////////////////////////////////////
//
// cu_save_loc
//
// Saves the current cursor row and
// column location
//

void cu_save_loc()
{
    VioGetCurPos(&c_row, &c_col, 0);
}

////////////////////////////////////////
//
// cu_rest_loc
//
// Restores a previously saved cursor
// location
//
```

```

void cu_rest_loc()
{
    VioSetCurPos(c_row, c_col, 0);
}

////////////////////////////////////
//
// cu_get_loc
//
// Saves the current cursor row and
// column location to parameter
// variables
//

void cu_get_loc(int *row, int *col)
{
    PUSHORT r, c;

    VioGetCurPos(r, c, 0);
    *row= (int)r;
    *col= (int)c;
}

////////////////////////////////////
//
// cu_move
//
// Moves the cursor to a specified
// row and column location
//

void cu_move( int row, int col)
{
    VioSetCurPos((USHORT)row, (USHORT)col, 0);
}

////////////////////////////////////
//
// cu_remove
//
// Removes the text cursor from the
// screen display
//

void cu_remove()
{
    G_CURSOR_VISIBLE= 0;
    VioGetCurType(&off_and_on, 0);
    off.yStart= 0;
    off.cEnd= 0;
    off.cx= 0;
    off.attr= 0xffff;
    VioSetCurType(&off, 0);
}

////////////////////////////////////
//
// cu_display
//

```

9-1 Continued.

```
// Displays the text cursor if it is
// not currently visible
//

void cu_display()
{
    if(!G_CURSOR_VISIBLE) {
        VioSetCurType( &off_and_on, 0 );
        G_CURSOR_VISIBLE= 1;
    }
}

////////////////////////////////////
//
// cu_set_size
//
// Sets the block text cursor size
//

void cu_set_size( int top, int bottom )
{
    USHORT t, b;

    t= (USHORT)top;
    b= (USHORT)bottom;
    VioGetCurType(&vioc_temp, 0);
    vioc_temp.yStart= t;
    vioc_temp.cEnd= b;
    VioSetCurType(&vioc_temp, 0);
}

////////////////////////////////////
//
// cu_save_size
//
// Saves the current cursor size to
// memory
//

void cu_save_size()
{
    VioGetCurType(&ssize_sr, 0);
}

////////////////////////////////////
//
// cu_rest_size
//
// Restores the size of a cursor
// whose shape had been previously
// saved
//

void cu_rest_size()
{
    VioSetCurType(&ssize_sr, 0);
}

////////////////////////////////////
//
```

```

// cu_get_shape
//
// Gets the current text cursor shape
// (as a 16-bit int)
//

int cu_get_shape()
{
int shape;

    VioGetCurType(&vioc_temp, 0);
    shape= (int)vioc_temp.cEnd;
    shape<= 8;
    shape|= (int)vioc_temp.yStart;
    return(shape);
}

////////////////////////////////////
//
// cu_set_shape
//
// Sets the cursor shape using a
// 16-bit int
//

void cu_set_shape(int shape)
{
    vioc_temp.yStart= (USHORT)shape;
    shape>= 8;
    vioc_temp.cEnd= (USHORT)shape;
    VioSetCurType(&vioc_temp, 0);
}

////////////////////////////////////
//
// bleep
//
// Dummy function
//

void bleep()
{

}

```

9-1 Ends.

10

OS/2 full screen management functions

This chapter presents the source code to the OS/2 screen management object module that has been placed in the OS2TEXT.LIB library file. Table 10-1 repeats the screen function prototypes.

Table 10-1 Screen function prototypes.

void	scrn_write(int row, int col, int len, char *str, UCHAR attr);
void	scrn_init(void);
int	scrn_read_char(unsigned short, unsigned short);
void	scrn_save(void);
void	scrn_restore(void);
void	scrn_clear(void);
void	scrn_attr(int, int, int, unsigned char);
void	scrn_change_attr(unsigned char);
void	scrn_char(int row, int col, char ch, UCHAR attr);
void	scrn_repeat_char(int row, int col, int len, char ch, UCHAR attr);
void	scrn_chr(int, int, int);

There are three source modules that all contribute to the library's screen management functionality. These three source files are MAKE.C (FIG. 10-1), RECT.C (FIG. 10-2) and SCREEN.C (FIG. 10-3).

Figure 10-1 presents the source code listing to MAKE.C. Functions presented in this source module facilitate the handling of screen attributes and tokens.

10-1 The source code listing to MAKE.C.

```

////////////////////////////////////
//
// make.c
//
// OS/2 version
//

//
// include file
//

#include "tproto.h"

////////////////////////////////////
//
// mk_attr
//
// Makes an attribute
//

UCHAR mk_attr(UCHAR fore,
              UCHAR back,
              UCHAR inten,
              UCHAR blink)
{
    UCHAR attr;
    attr= back << 4;
    return((UCHAR) attr | fore | inten | blink );
}

////////////////////////////////////
//
// mk_attr_intense
//
// Makes an existing attribute
// intense
//

UCHAR mk_attr_intense(UCHAR attr)
{
    return(attr | ON_INTENSITY);
}

////////////////////////////////////
//
// mk_attr_intense_blink
//
// Makes an existing attribute intense
// and blink
//

UCHAR mk_attr_intense_blink(UCHAR attr)

```

```

{
    return(attr | ON_INTENSITY | ON_BLINK);
}

////////////////////////////////////
//
// mk_attr_blink
//
// Makes an existing attribute blink
//

UCHAR mk_attr_blink(UCHAR attr)
{
    return(attr | ON_BLINK);
}

////////////////////////////////////
//
// mk_attr_inverse
//
// Makes an existing attribute
// inverse
//

UCHAR mk_attr_inverse(UCHAR attr)
{
    UCHAR u1,u2;
    u1= attr & 0xff;
    u2= u1;
    u2<<= 4;
    return (u2 | (u1 >> 4)) & 0x7f;
}

////////////////////////////////////
//
// mk_token
//
// Makes a token
//

int mk_token(UCHAR ch, UCHAR attr)
{
    int ret_val;
    ret_val= 0;
    ret_val= ch;
    return ((ret_val << 8) | attr);
}

////////////////////////////////////
//
// mk_char_attr
//
// Takes a screen token and returns an
// 8-bit char and 8-bit unsigned char
// attribute via pointers passed as
// parameters
//

```

10-1 Continued.

```
void mk_char_attr(int token, char *ch, UCHAR *attr)
{
    *attr= (UCHAR)token & 0x00ff;
    token>>= 8;
    *ch= (UCHAR) token;
}
```

10-1 Ends.

Figure 10-2 presents the source code listing to RECT.C. This source module presents functions that facilitate the management of rectangular portions of the screen.

10-2 The source code listing to RECT.C.

```
////////////////////////////////////
//
// rect.c
//
// OS/2 Version
//

#include <stdlib.h>
#include <stdio.h>
#include <malloc.h>
#include <string.h>

#include "tproto.h"

static
char buff32[80] = { 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32 };

////////////////////////////////////
//
// saveRect
//
// Saves a rectangular image to
// memory
//

void saveRect(RECT *R)
{
    unsigned int *iptr;
    int          row;
    int          column;

    iptr= (unsigned int *)R->image;
    for(row= R->ul_row; row <= R->lr_row; row++) {
        for(column= R->ul_col; column <= R->lr_col; column++) {
```

```

        *iptr++= scrn_read_char(row, column);
    }
}

////////////////////////////////////
//
// restRect
//
// Restores a previously saved
// rectangular image
//

void restRect(RECT *R)
{
    unsigned int *iptr;
    int          row, token;
    int          column;
    UCHAR        attr;
    char         ch;

    iptr= (unsigned int *) R->image;

    for(row= R->ul_row; row <= R->lr_row; row++) {
        for(column= R->ul_col; column <= R->lr_col; column++) {
            token= *iptr++;
            mk_char_attr(token, &ch, &attr);
            scrn_char(row, column, ch, attr);
        }
    }
}

////////////////////////////////////
//
// setRect
//
// Initializes a RECT structure
//

RECT *setRect(RECT *R, int ur, int uc, int lr, int lc)
{
    int size;
    R= (RECT *)malloc(sizeof(RECT));
    if(!R) {
        scrn_clear();
        cu_display();
        cu_move(0, 0);
        printf("Memory allocation Error: setRect\n");
        exit(0);
    }
    R->ul_row= ur;
    R->ul_col= uc;
    R->lr_row= lr;
    R->lr_col= lc;
    size= sizeRect(R);
    R->image= (unsigned int *)calloc(size, sizeof(int));
    return(R);
}

```

10-2 Continued.

```
////////////////////////////////////////
//
// sizeRect
//
// Calculates the size of a rectangular
// screen image
//

unsigned int sizeRect(RECT *R)
{
    int height, width, size;

    height= R->lr_row - R->ul_row + 1;
    width= R->lr_col - R->ul_col + 1;
    size= height * width;
    return(size);
}

////////////////////////////////////////
//
// dsyRect
//
// Destroys a rectangular structure
// by freeing previously allocated
// memory
//

void dsyRect(RECT *R)
{
    free((void *)R->image);
    free((void *)R);
}

////////////////////////////////////////
//
// subRect
//
// Subtracts a one RECT structure
// from another
//

void subRect( RECT *destR, RECT *srceR )
{
    destR->ul_row-= srceR->ul_row;
    destR->ul_col-= srceR->ul_col;
    destR->lr_row-= srceR->lr_row;
    destR->lr_col-= srceR->lr_col;
}

////////////////////////////////////////
//
// addRect
//
// Adds one RECT structure to another
//

void addRect(RECT *destR, RECT *srceR)
{

```

```

    destR->ul_row+= srceR->ul_row;
    destR->ul_col+= srceR->ul_col;
    destR->lr_row+= srceR->lr_row;
    destR->lr_col+= srceR->lr_col;
}

////////////////////////////////////
//
// dupRect
//
// Duplicates one RECT structure
// with another

void dupRect( RECT *destR, RECT *srceR)
{
    destR->ul_row= srceR->ul_row;
    destR->ul_col= srceR->ul_col;
    destR->lr_row= srceR->lr_row;
    destR->lr_col= srceR->lr_col;
}

////////////////////////////////////
//
// clrRect
//
// Clears a RECT structure using the
// NORMAL (7) screen attribute
//

void clrRect(RECT *R)
{
    int row;
    int column;
    int row_stop, col_stop;
    int token;

    row_stop= R->lr_row;
    col_stop= R->lr_col;

    for(row= R->ul_row; row < row_stop; row++) {
        for(column= R->ul_col; column < col_stop; column++) {
            scrn_char( row, column, ' ', 7 );
        }
    }
}

////////////////////////////////////
//
// offRect
//
// Shifts the row and column locations
// within a RECT structure by the
// offset values.
//

void offRect( RECT *destR, int row_off, int col_off )
{
    destR->ul_row+= row_off;

```

10-2 Continued.

```
destR->ul_col+= col_off;
destR->lr_row+= row_off;
destR->lr_col+= col_off;
}

////////////////////////////////////
//
// boxRect
//
// Places a border around a rectangle
//

void boxRect(RECT *R, int box_type, unsigned char attr)
{
int row,column;
int token;
int top_bot, left_right, ul, ur, ll, lr; // box choices

switch(box_type)
{
case 1:
top_bot= 196;
left_right= 186;
ul= 214;
ur= 183;
ll= 211;
lr= 189;
break;
case 2:
top_bot= 205;
left_right= 179;
ul= 213;
ur= 184;
ll= 212;
lr= 190;
break;
case 3:
top_bot= 205;
left_right= 186;
ul= 201;
ur= 187;
ll= 200;
lr= 188;
break;
default:
top_bot= 196;
left_right= 179;
ul= 218;
ur= 191;
ll= 192;
lr= 217;
break;
}

for(row= R->ul_row; row < R->lr_row; row++) {
scrn_write(row, R->ul_col, R->lr_col - R->ul_col, buff32, attr);
}

// draw top and bottom
```



```

for(column= R->ul_col; column < R->lr_col; ++column) {
    scrn_char(R->ul_row, column, top_bot, attr);
    scrn_char(R->lr_row, column, top_bot, attr);
}

// draw left and right borders

for(row= R->ul_row; row < R->lr_row; ++row) {
    scrn_char(row, R->ul_col, left_right, attr);
    scrn_char(row, R->lr_col, left_right, attr);
}

// plop the four corners

scrn_char(R->ul_row, R->ul_col, ul, attr);

scrn_char(R->ul_row, R->lr_col, ur, attr);

scrn_char(R->lr_row, R->ul_col, ll, attr);

scrn_char(R->lr_row, R->lr_col, lr, attr);
}

```

10-2 Ends.

Figure 10-3 presents the OS/2 based source code listing to SCREEN.C. This source file uses OS/2's VIO functions as building blocks for the migration library's higher level screen management routines.

10-3 The source code listing to SCREEN.C.

```

////////////////////////////////////
//
// screen.c
//
// OS/2 Version
//
////////////////////////////////////
//
// defines required for OS/2
//

#define INCL_VIO
#define INCL_KBD
#define INCL_MOU
#define INCL_DOSPROCESS
#define INCL_DOSSEMAPHORES

////////////////////////////////////
//
// include files
//

#include <os2.h>
#include <string.h>
#include <ctype.h>

#include "tproto.h"

```

10-3 Continued.

```
////////////////////////////////////
//
// static buffers
//

static UCHAR scrn_buf[(80 * 25) * 2];
static char buff32[80]= { 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                          32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                          32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                          32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                          32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                          32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                          32, 32, 32, 32, 32, 32, 32, 32, 32, 32 };

////////////////////////////////////
//
// scrn_save
//
// Saves the screen image to memory
//

void scrn_save()
{
    HVIO    hvio= 0;
    UCHAR   *cptr;
    USHORT  row;
    USHORT  column;
    USHORT  length;

    cptr= (unsigned char *)scrn_buf;
    length= 160;

    for(row= 0 ; row < 25; row++) {
        VioReadCellStr(cptr, &length, row, 0, hvio);
        cptr+= length;
    }
}

////////////////////////////////////
//
// scrn_restore
//
// Restore a screen image that had
// been previously saved
//

void scrn_restore()
{
    HVIO    hvio= 0;
    UCHAR   *cptr;
    USHORT  row;
    USHORT  column;
    USHORT  length;

    cptr= (unsigned char *)scrn_buf;
    length= 160;
```

```

        for(row= 0; row < 160; row++) {
            VioWrtCellStr(cptr, length, row, 0, hvio);
            cptr+= length;
        }

    }

    //////////////////////////////////////
    //
    // scrn_clear
    //
    // Clear the screen using the NORMAL
    // (7) attribute
    //
    void scrn_clear()
    {
        int row;

        for(row= 0; row < 25; row ++) {
            scrn_write(row, 0, 80, buff32, 7);
        }
    }

    //////////////////////////////////////
    //
    // scrn_change_attr
    //
    // Changes the attribute for the
    // entire screen without altering the
    // characters
    //
    void scrn_change_attr(unsigned char attr)
    {
        int row;

        for(row= 0; row < 25; row ++) {
            scrn_attr(row, 0, 80, attr);
        }
    }

    //////////////////////////////////////
    //
    // scrn_chr
    //
    // Writes a screen token (char +
    // (256 * attr)) to a specified row and
    // column screen location
    //
    void scrn_chr(int row, int col, int token)
    {
        USHORT r, c;
        UCHAR at;
        char ch;

        r= (USHORT)row;
        c= (USHORT)col;

```

10-3 Continued.

```
    ch= (char)token;
    token= token >> 8;
    at= (UCHAR)token;
    scrn_char(row, col, ch, at);
}

////////////////////////////////////
//
// scrn_repeat_char
//
// Repeats a character on a row to the
// screen a specified number of times
//

void scrn_repeat_char(int row, int col, int len, char ch, UCHAR attr)
{
    int counter;

    for(counter= 0; counter < len; counter++) {
        scrn_char(row, col++, ch, attr);
    }
}

////////////////////////////////////
//
// vdBox
//
// Internal function that draws a
// lined border around a rectangle
//

void vdBox(RECT *R, int box_type, unsigned char attr)
{
    int row, column;
    int top_bot, left_right, ul, ur, ll, lr;

    switch(box_type)
    {
        case 1:
            top_bot = 196;
            left_right = 186;
            ul = 214;
            ur = 183;
            ll = 211;
            lr = 189;
            break;
        case 2:
            top_bot = 205;
            left_right = 179;
            ul = 213;
            ur = 184;
            ll = 212;
            lr = 190;
            break;
        case 3:
            top_bot = 205;
            left_right = 186;
    }
```

```

        ul = 201;
        ur = 187;
        ll = 200;
        lr = 188;
        break;
    default:
        top_bot = 196;
        left_right = 179;
        ul = 218;
        ur = 191;
        ll = 192;
        lr = 217;
        break;
    }

    for(row= R->ul_row; row < R->lr_row; row++) {
        scrn_write(row, R->ul_col, R->lr_col - R->ul_col, buff32, attr);
    }

    // draw top and bottom

    for(column= R->ul_col; column < R->lr_col; ++column) {
        scrn_char(R->ul_row, column, top_bot, attr);
        scrn_char(R->lr_row, column, top_bot, attr);
    }

    // draw left and right borders

    for(row= R->ul_row; row < R->lr_row; ++row) {
        scrn_char(row, R->ul_col, left_right, attr);
        scrn_char(row, R->lr_col, left_right, attr);
    }

    // plop the four corners

    scrn_char(R->ul_row, R->ul_col, ul, attr);
    scrn_char(R->ul_row, R->lr_col, ur, attr);
    scrn_char(R->lr_row, R->ul_col, ll, attr);
    scrn_char(R->lr_row, R->lr_col, lr, attr);
}

////////////////////////////////////////
//
// scrn_write
//
// Writes a string to the screen using
// a specified attribute
//

void scrn_write(int row, int col, int len, char *str, UCHAR attr)
{
    HVIO hvio= 0;
    unsigned char bAttr;

    bAttr= attr;

    if(!len) {
        len= strlen(str);
    }

```

10-3 Continued.

```
    }

    VioWrtCharStrAtt( str,
                      len,
                      row,
                      col,
                      &bAttr,
                      hvio );
}

////////////////////////////////////
//
// scrn_char
//
// Writes a character to the screen
// at a specified row and column
// location
//
void scrn_char(int row, int col, char ch, UCHAR attr)
{
    HVIO hvio= 0;
    UCHAR bAttr;

    bAttr= attr;

    VioWrtCharStrAtt( &ch,
                      1,
                      row,
                      col,
                      &bAttr,
                      hvio );
}

////////////////////////////////////
//
// scrn_attr
//
// This function alters a screen
// attribute without altering the
// character associated with that
// attribute
//
void scrn_attr(int row, int col, int len, unsigned char attr)
{
    HVIO hvio= 0;
    UCHAR bAttr;

    bAttr= attr;

    VioWrtNAttr( &bAttr,
                 len,
                 row,
                 col,
                 hvio );
}
```

```

////////////////////////////////////
//
// scrn_read_char
//
// Reads a character and attribute
// from a specified row and column
// screen location
//

int scrn_read_char(USHORT row, USHORT col)
{
    HVIO    hvio= 0;
    char    buffer[2];
    PCH     str;
    USHORT  length= 2;

    str= buffer;

    VioReadCellStr(str, &length, (USHORT)row, (USHORT)col, hvio);

    return(mk_token(*str, *(str + 1)));
}

////////////////////////////////////
//
// scrn_init
//
// Empty function for OS/2 library
// included to facilitate migration
// from DOS to OS/2
//

void scrn_init()
{
}

////////////////////////////////////
//
// rdImg
//
// Reads a rectangular portion of a
// screen defined within the
// WIND structure to memory
//

void rdImg(WIND *R)
{
    HVIO    hvio= 0;
    UCHAR   *cptr;
    USHORT  row;
    USHORT  column;
    USHORT  length;

    cptr= (UCHAR *)R->img_ptr;
    length= (R->lr_col - R->ul_col) + 1;
    length*= 2;

    for(row= R->ul_row; row <= R->lr_row; row++) {

```

10-3 Continued.

```
VioReadCellStr(cpctr, &length, row, R->ul_col, hvio);
cpctr+= (length * 2);
}

}

////////////////////////////////////
//
// wrImg
//
// Writes a rectangular portion of a
// screen defined within the
// WIND structure to the screen
//

void wrImg(WIND *R)
{
    HVIO    hvio= 0;
    UCHAR   *cpctr;
    USHORT  row;
    USHORT  column;
    USHORT  length;

    cpctr= (unsigned char *)R->img_ptr;
    length= (R->lr_col - R->ul_col) + 1;
    length*= 2;

    for(row= R->ul_row; row <= R->lr_row; row++) {
        VioWrtCellStr(cpctr, length, row, R->ul_col, hvio);
        cpctr+= (length * 2);
    }
}

}

////////////////////////////////////
//
// rdWind
//
// Reads a rectangular portion of a
// screen defined within the
// WIND structure to memory
//

void rdWind(WIND *R)
{
    HVIO    hvio= 0;
    UCHAR   *cpctr;
    USHORT  row;
    USHORT  column;
    USHORT  length;

    cpctr= (UCHAR *)R->wind_ptr;
    length= (R->lr_col - R->ul_col) + 1;
    length *= 2;

    for(row= R->ul_row; row <= R->lr_row; row++) {
        VioReadCellStr(cpctr, &length, row, R->ul_col, hvio);
```



```

        cptr+= (length * 2);
    }

}

////////////////////////////////////
//
// wrWind
//
// Writes a rectangular portion of a
// screen defined within the
// WIND structure to the screen
//

void wrWind(WIND *R)
{
    HVIO    hvio= 0;
    UCHAR   *cptr;
    USHORT  row;
    USHORT  column;
    USHORT  length;

    cptr= (UCHAR *)R->wind_ptr;
    length= (R->lr_col - R->ul_col) + 1;
    length *= 2;

    for(row= R->ul_row; row <= R->lr_row; row++) {
        VioWrtCellStr(cptr, length, row, R->ul_col, hvio);
        cptr+= (length * 2);
    }
}

////////////////////////////////////
//
// wrBox
//
// Internal function that draws a
// lined border around a rectangle
//

void wrBox(WIND *R)
{
    int row,column;
    int token;
    int top_bot, left_right, ul, ur, ll, lr;

    switch(R->box_type)
    {
        case 1:
            top_bot = 196;
            left_right = 186;
            ul = 214;
            ur = 183;
            ll = 211;
            lr = 189;

```

10-3 Continued.

```
        break;
    case 2:
        top_bot = 205;
        left_right = 179;
        ul = 213;
        ur = 184;
        ll = 212;
        lr = 190;
        break;
    case 3:
        top_bot = 205;
        left_right = 186;
        ul = 201;
        ur = 187;
        ll = 200;
        lr = 188;
        break;
    default:
        top_bot = 196;
        left_right = 179;
        ul = 218;
        ur = 191;
        ll = 192;
        lr = 217;
        break;
}

token = mk_token((int)' ', R->attr);

for(row= R->ul_row; row < R->lr_row; row++) {
    scrn_write(row, R->ul_col, R->lr_col - R->ul_col, buff32, R->attr);
}

// draw top and bottom

for(column= R->ul_col; column < R->lr_col; ++column) {
    scrn_char(R->ul_row, column, top_bot, R->attr);
    scrn_char(R->lr_row, column, top_bot, R->attr);
}

// draw left and right borders

for(row= R->ul_row; row < R->lr_row; ++row) {
    scrn_char(row, R->ul_col, left_right, R->attr);
    scrn_char(row, R->lr_col, left_right, R->attr);
}

// plop the four corners

scrn_char(R->ul_row, R->ul_col, ul, R->attr);

scrn_char(R->ul_row, R->lr_col, ur, R->attr);

scrn_char(R->lr_row, R->ul_col, ll, R->attr);

scrn_char(R->lr_row, R->lr_col, lr, R->attr);
}
```

```

////////////////////////////////////////
//
// sizeImg
//
// Calculates the size of the memory
// required to hold a window image
//

unsigned int sizeImg(WIND *R)
{
    int height, width, size;

    height= R->lr_row - R->ul_row;
    width= R->lr_col - R->ul_col;
    ++height;
    ++width;
    size= height * width;
    return(size);
}

```

10-3 Ends.

11

OS/2 mouse management functions

This chapter presents the source code to the OS/2 mouse management object module that has been placed in the OS2TEXT.LIB library file. Table 11-1 repeats the mouse function prototypes.

Table 11-1 High level mouse function prototypes.

int	ms_init(void);
void	ms_on(void);
void	ms_off(void);
int	ms_status(int *x, int *y);
void	ms_map_display(int row, int col, int key_val);

Figure 11-1 presents the OS/2 based source code listing to MOUSE.C. This function uses OS/2's MOUS functions that reflect object modules contained in the OS2TEXT.LIB library file.

11-1 The source code listing to MOUSE.C.

```
////////////////////////////////////  
//  
// mouse.c  
//  
// OS/2 Version  
//  
////////////////////////////////////
```

11-1 Continued.

```
////////////////////////////////////
//
// OS/2 defines here
//

#define INCL_VIO
#define INCL_KBD
#define INCL_MOU
#define INCL_DOSPROCESS
#define INCL_DOSSEMAPHORES

////////////////////////////////////
//
// includes here
//

#include <os2.h>
#include "tproto.h"

////////////////////////////////////
//
// global variables
//

MOUEVENTINFO mouev;
HMOU          mouse_handle;

////////////////////////////////////
//
// ms_init
//
// Initialize the mouse for use
//

int ms_init()
{
    USHORT buttons= 0;

    MouOpen(NULL, &mouse_handle);
    MouGetNumButtons( &buttons, mouse_handle);
    return buttons;
}

////////////////////////////////////
//
// ms_off
//
// Turn off the mouse
//

void ms_off()
{
    NOPTRRECT mourt= {0,0,24,79};
    MouRemovePtr( &mourt, mouse_handle );
}

////////////////////////////////////
//
```

```

// ms_on
//
// turn on the mouse
//

void ms_on()
{
    USHORT status= 0;

    MouDrawPtr( mouse_handle );
}

////////////////////////////////////
//
// ms_status
//
// Reports button press status and
// the row and column location of the
// mouse. The information is only
// reported on a button press event.
//

int ms_status(int *x, int *y)
{
    USHORT fWait = MOU_NOWAIT;

    MouReadEventQue(&mouev, &fWait, mouse_handle);

    if(mouev.time) {
        if(mouev.fs & MOUSE_BN1_DOWN ) {
            *y= mouev.row * 8;
            *x= mouev.col * 8;
            if( ((int)mouev.fs == 4) || ((int)mouev.fs == 2) ) {
                MouFlushQue(mouse_handle);
                return(1);
            }
            else
                return((int)0);
        }
        if(mouev.fs & MOUSE_BN2_DOWN ) {
            *y= mouev.row * 8;
            *x= mouev.col * 8;
            if( ((int)mouev.fs == 16) || ((int)mouev.fs == 8) ) {
                MouFlushQue(mouse_handle);
                return(2);
            }
            else
                return((int)0);
        }
    }
    else {
        return((int)0);
    }
    return ((int)0);
}

////////////////////////////////////
//
// ms_map_display
//

```

11-1 Continued.

```
// Reports the mouse location on the
// screen when there is a button press.
// This function will prove invaluable
// in creating dialog boxes and
// menus.
//

void ms_map_display(int row, int col, int key_val)
{
    UCHAR  exit_flag= 0;
    char   buffer[20];
    int    key;
    int    x= 0, y= 0;

    do {

        key= kb_status();

        if(!key) {
            key= ms_status(&x, &y);
            if(key == 1) {
                scrn_write(row, col, 0, "Left Button Press ", 7);
            }
            if(key == 2) {
                scrn_write(row, col, 0, "Right Button Press", 7);
            }
            memset(buffer, 0, 20);
            sprintf(buffer, "x= %03d  y= %03d", x, y);
            scrn_write(row + 1, col, 0, buffer, 7);
        }

        if(key == key_val) {
            exit_flag= 1;
        }

    } while(!exit_flag);
}
```

11-1 Ends.

12

OS/2 full screen window management functions

This chapter presents the source code to the OS/2 window management object module that has been placed in the OS2TEXT.LIB library file. Table 12-1 repeats the high level window function prototypes.

Table 12-1 High Level window function prototypes.

WIND	*wind_init(WIND *W_PTR, int ulr, int ulc, int lrr, int lrc, UCHAR attr, int border, char *title);
int	wind_kb_edit(WIND *W, char *response, int row, int column, int dlen, int opt, UCHAR attr);
void	wind_display(WIND *);
void	wind_remove(WIND *);
void	wind_attr(WIND *, int, int, int, unsigned char);
void	wind_write(WIND *, int, int , int , char *, UCHAR);

Table 12-1 Continued.

void	wind_char(WIND *, int row, int col, char ch, UCHAR attr);
void	wind_repeat_char(WIND *, int row, int col, int len, char ch, UCHAR attr);
void	wind_destroy(WIND *);
int	wind_read_char(WIND *, int, int);
void	wind_clear(void);
void	wind_cu_move(WIND *, int, int);

Figure 12-1 presents the OS/2 based source code listing to WINDOW.C. This source file uses functions presented in RECT.C, MAKE.C, and SCREEN.C as building blocks for the migration library's higher level window management routines.

12-1 The source code listing to WINDOW.C.

```
////////////////////////////////////
//
// window.c
//
// OS/2 version
//
////////////////////////////////////

////////////////////////////////////
//
// includes and defines here
//

#include <stdlib.h>
#include <malloc.h>
#include <string.h>
#include <stdio.h>

#include "tproto.h"

#define wNULL 0
#define W_SIZE sizeof(WIND)
extern unsigned int sizeImg(WIND *);

////////////////////////////////////
//
// wind_destroy
//
// Destroys a window structure by
// freeing previously dynamically
// allocated memory
//

void wind_destroy(WIND *W)
{
    if(W->img_ptr != wNULL) {
```

```

        free((void *)W->img_ptr);
    }

    if(W->wind_ptr != wNULL) {
        free((void *)W->wind_ptr);
    }

    if(W->t_title != wNULL) {
        free((void *)W->t_title);
    }

    if(W != wNULL) {
        free((void *)W);
    }
}

////////////////////////////////////
//
// strt_wind
//
// This function is called the first
// time that a window is displayed
//

void strtWind(WIND *W)
{
    char *tptr;

    tptr= W->t_title;

    rdImg(W);

    wrBox(W);

    rdWind(W);

    W->visible= 1;

    if(W->show_top) {
        wind_write(W, 0, W->top_offset, W->top_length, tptr, W->attr);
    }
}

////////////////////////////////////
//
// setWind
//
// Preps the window structure
//

WIND *setWind(WIND *W, int ul_row, int ul_col, int lr_row, int lr_col)
{
    W= (WIND *)calloc(W_SIZE, sizeof(char));
    if(!W) {
        scrn_clear();
        cu_display();
    }
}

```

12-1 Continued.

```
        cu_move(0, 0);
        printf("Memory allocation Error: setWind\n");
        exit(0);
    }

    W->ul_row= ul_row;
    W->ul_col= ul_col;
    W->lr_row= lr_row;
    W->lr_col= lr_col;
    W->img_size= sizeImg(W);
    W->img_ptr= (unsigned int *)calloc(W->img_size, sizeof(int));
    if(!W->img_ptr) {
        scrn_clear();
        cu_display();
        cu_move(0, 0);
        printf("Memory allocation Error: W->img_ptr\n");
        exit(0);
    }

    W->wind_ptr = (unsigned int *)calloc(W->img_size, sizeof(int));
    if(!W->wind_ptr) {
        scrn_clear();
        cu_display();
        cu_move(0, 0);
        printf("Memory allocation Error: setWect\n");
        exit(0);
    }

    W->visible= aFALSE;
    W->box_type =0;
    W->attr= NORMAL;
    W->t_title= 0;
    W->b_title= 0;
    W->show_top= aFALSE;
    W->show_bot= aFALSE;
    return(W);
}

////////////////////////////////////////
//
// setBord
//
// Sets the border variable in the
// window structure
//
// set the border
//
// #   T B L W
// - - - - -
// 0 = S_S_S_S
// 1 = S_S_D_D
// 2 = D_D_S_S
// 3 = D_D_D_D
//
//

void setBord(WIND *W, int type)
{
    W->box_type= type;
```

```

}

////////////////////////////////////
//
// wind_attr
//
// Sets a series of attributes on a row
// in a window
//

void wind_attr(WIND *W, int row, int col, int length, UCHAR attr)
{
    row+= W->ul_row;
    col+= W->ul_col;
    scrn_attr( row, col, length, attr);
}

////////////////////////////////////
//
// wind_remove
//
// Removes a window from the screen
// by restoring the screen that
// had previously been under the window
// area
//

void wind_remove(WIND *W)
{
    if(W->visible) {
        rdWind(W);
        wrImg(W);
        W->visible= 0;
    }
}

////////////////////////////////////
//
// setAttr
//
// Sets the attribute in the window
// structure
//

void setAttr(WIND *W, UCHAR attr)
{
    W->attr= attr;
}

void
wind_display(W)
WIND *W;
{
    if(!W->visible)
    {
        rdImg(W);
        wrWind(W);
    }
}

```

12-1 Continued.

```
W->visible = 1;
}
}

////////////////////////////////////
//
// setTitle
//
// Prepares the window structure
// to receive the title string and
// title location
//

void setTitle(WIND *W, char *top)
{
    W->top_length= strlen(top);
    W->top_offset= ( (W->lr_col - W->ul_col) - W->top_length )/2;
    W->top_offset+= 1;
    W->t_title= (char *)malloc(W->top_length + 1);
    if(!W->t_title) {
        scrn_clear();
        cu_display();
        cu_move(0, 0);
        printf("Memory allocation Error: setTitle\n");
        exit(0);
    }
    memset(W->t_title, '\0', W->top_length + 1);
    strcpy(W->t_title, top);
    W->show_top= aTRUE;
}

////////////////////////////////////
//
// wind_write
//
// Writes a character string to the
// window
//

void wind_write(WIND *W,
                int row,
                int col,
                int length,
                char *str,
                UCHAR attr)
{
    int count;

    if(!length) {
        length= strlen(str);
    }

    if(col==CENTER) {
        col=( W->lr_col - W->ul_col - length-1) / 2;
    }
    row+= W->ul_row;
    col+= W->ul_col;

    scrn_write( row, col, length, str, attr);
}
```

```

}

////////////////////////////////////
//
// wind_init
//
// Handles all of the mid-level window
// initialization functions
//

WIND *wind_init(WIND *W_PTW,
                int ulr,
                int ulc,
                int lrr,
                int lrc,
                unsigned char attr,
                int border,
                char *title )
{
    // Allocate memory and return pointer to structure

    W_PTW = setWind(W_PTW,ulr,ulc,lrr,lrc);

    // Set Window Attr - Fore,Back,Intensity,Blink

    setAttr(W_PTW, attr);

    // Set Window Border

    setBord(W_PTW,border);

    // Set the bottom title

    setTitle(W_PTW,title);

    // Display window

    strtWind(W_PTW);

    // return the pointer

    return W_PTW;
}

////////////////////////////////////
//
// wind_kb_edit
//
// Edit an alphanumeric field in a
// window
//

int wind_kb_edit(WIND *W,
                 char *response,
                 int row,
                 int column,
                 int dlen,
                 int opt,
                 UCHAR attr)
{

```

12-1 Continued.

```

int ret_val;

    ret_val= kb_edit(response,
                    row + W->ul_row,
                    column + W->ul_col,
                    dlen,
                    opt,
                    attr);
    return(ret_val);
}

////////////////////////////////////
//
// wind_repeat_char
//
// Repeats a character on a row in a
// window
//

void wind_repeat_char(WIND *W,
                    int row,
                    int col,
                    int len,
                    char ch,
                    UCHAR attr)
{
    scrn_repeat_char(W->ul_row + row,
                    W->ul_col + col,
                    len,
                    ch,
                    attr);
}

////////////////////////////////////
//
// wind_char
//
// Writes a character to a window
// location
//

void wind_char(WIND *W, int row, int col, char ch, UCHAR attr)
{
    scrn_char(W->ul_row + row,
              W->ul_col + col,
              ch,
              attr);
}

```

12-1 Ends.

13

OS/2 printer management functions

In OS/2 you can treat the printer as a device in a similar fashion to that of a disk drive. The only operations that are supported by this migration library when using the printer as a device are opening, writing to, and closing. These foundation operations have been used to create some higher level printer management functions.

This chapter presents the source code to the OS/2 printer management object module that has been placed in the OS2TEXT.LIB library file. Table 13-1 repeats all the printer function prototypes.

**Table 13-1 Printer
function prototypes.**

int	print_open(int num);
int	print_close(int num);
int	print_newline(int num);
int	print_cr(int num);
int	print_string(int num, char *);
int	print_char(int, char);
int	print_scrn(int);
int	print_scrnFF(int);
int	print_status(int);
void	print_set_column(int, int);

Figure 13-1 presents the OS/2 based source code listing to PRINTER.C. This function uses OS/2's MOUS functions that reflect object modules contained in the OS2TEXT.LIB library file.

13-1 The source code listing to PRINTER.C.

```

////////////////////////////////////
//
// printer.c
//
// OS/2 Version
//
////////////////////////////////////

////////////////////////////////////
//
// OS/2 defines
//

#define INCL_DOSDEVICES
#define INCL_DOSDEVIOTCL
#define INCL_VIO
#define INCL_KBD
#define INCL_MOU
#define INCL_DOSPROCESS
#define INCL_DOSSEMAPHORES

////////////////////////////////////
//
// include files
//

#include <os2.h>
#include <stdio.h>
#include <string.h>
#include <memory.h>
#include <malloc.h>

#include "tproto.h"

static HFILE printer_handle;

////////////////////////////////////
//
// print_string
//
// Sends a string of bytes to the
// printer
//

int print_string(int num, char *buffer)
{
    ULONG action, wrote;
    APIRET rc= 0;

    rc= DosWrite(printer_handle,
                  (PVOID)buffer,
                  (ULONG)strlen(buffer),
                  &wrote);

```

```

        return((int)rc);
    }

    //////////////////////////////////////
    //
    // print_char
    //
    // Send one byte to the printer
    //

    int print_char(int num, char ch)
    {
        ULONG action, wrote;
        APIRET rc= 0;
        char c_val;

        c_val= ch;
        rc= DosWrite(printer_handle,
                     &c_val,
                     (ULONG)0x01,
                     &wrote);

        return((int)rc);
    }

    //////////////////////////////////////
    //
    // print_newline
    //
    // Send a new line byte sequence
    // to the printer
    //

    int print_newline(int num)
    {
        return((int)print_char(num, '\n'));
    }

    //////////////////////////////////////
    //
    // print_cr
    //
    // Send a carriage return byte to the
    // printer
    //

    int print_cr(int num)
    {
        return((int)print_char(num, aCR));
    }

    //////////////////////////////////////
    //
    // print_open
    //
    // Open the printer as a device that
    // may be written to
    //

    int print_open(int num)

```

13-1 Continued.

```
{
ULONG  action, wrote;
APIRET rc= 0, rc1= 0;
char   *pname[4] = { "LPT1",
                      "LPT2",
                      "LPT3",
                      "LPT4" };

char   *name;
char   SP_BS[2]= { 0x11, 8 };

    action= 2;

    name= pname[num];

    rc= DosOpen(name,
                &printer_handle,
                &action,
                0,
                0x20,
                0x01 | 0x10,
                0x01 | 0x10,
                0L);

    return((int)rc);
}

////////////////////////////////////
//
// print_close
//
// Close the printer device. This
// call also flushes the print
// buffer
//

int print_close(int num)
{
    return(DosClose(printer_handle));
}

////////////////////////////////////
//
// print_set_column
//
// Sets the printer head to a
// specified column. This function
// will prove useful in creating
// formatted printer output.
//

void print_set_column(int num, int column)
{
    int ctr;

    print_char(num, aCR);

    for(ctr= 0; ctr < column; ctr++) {
        print_char(num, ' ');
    }
}
```

13-1 Ends.

Part Three

DOS character mode library

Nine source code modules make up the DOS Library. Keyboard and mouse access are handled via the PC's BIOS. The screen is accessed via direct memory. All but one of the modules is written in C. The lone assembly module (FIG. 14-1) is KB_STAT.ASM.

Note that the C source files for the DOS library have the same names and the OS/2 library. For that reason, always make sure that you keep your OS/2 and DOS C source modules in separate directories.

14

DOS keyboard management functions

This chapter presents the source code to the DOS keyboard management object module that has been placed in the DOSTEXT.LIB library file. Table 14-1 repeats the keyboard function prototypes.

**Table 14-1 Keyboard
function prototypes.**

int	kb_edit(char *response, int row, int column, int dlen, int opt, UCHAR attr);
int	kb_read(void);
int	kb_status(void);
char	kb_char(void);
UCHAR	kb_scan(void);

Figure 14-1 presents the assembly listing for KB_STAT.ASM. This simple assembly file may be assembled using either MASM or TASM. I chose to code this file in assembly because program flow decisions needed to be made according to flag status. Writing the code in assembly proved trivial where writing it in C would have proven a tad more cumbersome.

Writing this file in assembly will also ease the task of bringing the migration library over to another DOS C compiler.

14-1 The source code listing to KB_STAT.ASM.

```
;/;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;/
;
; Name          gt_stat
;
; Synopsis      ret = kb_status()
;
;               int  0 -> on no key waiting
;               else key scan & char code
;
; Description   0 on key waiting and scan & char on key waiting
;
;/;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;/

        DOSSEG

        .MODEL LARGE,C

        .CODE

        public  kb_status

kb_status proc
        mov     AH,1      ; kb stat function
        int     16H       ; keybd int
        jnz     yeskey    ; jmp on no key waiting
        mov     AX,0       ; no key wait return 0
        jmp     keyexit
yeskey:
        mov     AH,0
        int     16H
keyexit:
        ret
kb_status endp
        end
```

14-1 Ends.

Figure 14-2 presents the DOS based source code listing to KEYBOARD.C. This source file uses DOS's BIOS functions as building blocks for the migration library's higher level keyboard routines.

14-2 The source code listing to KEYBOARD.C.

```
;/;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;/
//
// keyboard.c
//
// DOS Version
//
;/;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;/

;/;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;/
//
// include files here
```



```

//

#include <dos.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#include "tproto.h"

void bleep(void);

////////////////////////////////////
//
// kb_read
//
// Stop program execution and wait
// for a key press. The scan and
// character codes are returned
//

int kb_read()
{
    union REGS ir, or;

    // get key press via BIOS

    ir.h.ah= 0x00;

    // invoke BIOS

    int86(0x16, &ir, &or);

    return(or.x.ax);
}

////////////////////////////////////
//
// kb_char
//
// Stop program execution and wait
// for a key press. The char code
// is returned
//

char kb_char()
{
    union REGS ir, or;

    // get key press via BIOS

    ir.h.ah= 0x00;

    // invoke BIOS

    int86(0x16, &ir, &or);

    return(or.h.al);
}

```

14-2 Continued.

```
////////////////////////////////////////
//
// kb_scan
//
// Stop program execution and wait
// for a key press. The scan code
// is returned
//

UCHAR kb_scan()
{
    union REGS ir, or;

    // get key press via BIOS

    ir.h.ah= 0x00;

    // invoke BIOS

    int86(0x16, &ir, &or);

    return(or.h.ah);
}

////////////////////////////////////////
//
// kb_edit
//
// Alphanumeric field entry routine
//

int kb_edit(char *response,
            int row,
            int column,
            int dlen,
            int opt,
            UCHAR attr)
{
    int key;
    int start, stop;
    char *rptr;
    int i;
    int ins=0;
    char buf[80];
    int cur, start_column;
    int ret_val;
    int tlen;

    // set start column for stopper on left arrow

    start_column= column;

    // save cursor shape and location

    cur= cu_get_shape();
    cu_save_loc();

    // turn the cursor on
```

```

cu_display();

// place '\0' at end point of response buffer
*(response + dlen)= 0;

// make response string upper or lower
switch(opt)
{
    case LOWER:
        strlwr(response);
        break;
    case UPPER:
       strupr(response);
        break;
    case NAME:
        response= strlwr(response);
        *response= toupper(*response);
        break;
}

// copy contents of response buffer to buf[]
for(i= 0; i < dlen; i++) {
    buf[i]= response[i];
}

// set start and stop variables
start= column;
stop= start + dlen;

// alter screen attributes for edit field
scrn_attr(row, column, dlen, attr);

// if *response != 0 then write string to screen
if(*response) {
    scrn_write(row, column, 0, response, attr);
}

// adjust cursor location to response string end
cu_move(row, column+= strlen(response));

// set response pointer to end of response buffer
rptr= response + strlen(response);

// wait for key press
key= kb_read();

// process key press first time through
switch(key)
{
    case ESCAPE:
    case ENTER:

```

14-2 Continued.

```
case F1:
case F2:
case F3:
case F4:
case F5:
case F6:
case F7:
case F8:
case F9:
case F10:
case UP_ARROW:
case DOWN_ARROW:
case UP_ARROW_K:
case DOWN_ARROW_K:
case TAB:
    cu_set_shape(cur); // restore cursor shape
    cu_rest_loc();     // restore cursor location
    return(key);       // return key press value

case HOME:
case PGUP:
    memset(response, 0, dlen + 1); // clear response buffer
    rptr= response;                // reset pointer
    scrn_write(row, start, dlen, response, attr); // clr scrn
    column= start;                 // reset column to start
    cu_move(row, column);          // adjust cursor location
    break;

case CNTL_LEFTA:
    while(*--rptr!=' ');           // mv ptr to ' ' char
    rptr++;                        // and incr ptr by 1

    // adjust cursor if ' ' is after start

    if(start_column<start+(int)(rptr-response)) {
        cu_move(row, column= start + (int)(rptr - response));
    }
    else {
        // set column & adjust cursor

        column= start_column;
        cu_move(row, start_column);
    }

    // write response buffer to screen

    scrn_write(row, start, dlen, response, attr);
    break;

case LEFT_ARROW:
case LEFT_ARROW_K:
    // is cursor right of start?

    if(start_column < column) {
        // yes -> adjust cursor left

        cu_move(row, --column);

        // adjust ptr
```

```

        rptr--;
    }

case END:
case INSERT:
case DELETE:
case RIGHT_ARROW:
case RIGHT_ARROW_K:
case PGDN:

    // write response buffer to screen

    scrn_write(row, start, dlen, response, attr);
    break;

default:

    // NULL out scan code

    key&= 0x00ff;

    // set letter case

    switch(opt)
    {
        case LOWER:
            key= tolower(key);
            break;

        case UPPER:
            key= toupper(key);
            break;
    }

    // if printable character

    if((key >= 0x20) && (key <= 0x7e)) {
        // NULL buffer

        memset(response, 0, dlen + 1);

        // set pointer to response start

        rptr= response;

        // place key in response

        *rptr++= (char)key;

        // write buffer to screen

        scrn_write(row, start, dlen, response, attr);

        // adjust column pointer

        column= start + 1;

        // adjust cursor position

        cu_move(row, column);
    }

```

14-2 Continued.

```
        // is the key a backspace?

        if(key == aBS) {
            // if column is greater than start

            if(column > start) {
                // backspace in response buffer

                rptr--;

                // place NULL at new location

                *rptr= 0;

                // write response buffer

                scrn_write(row, start, dlen, response, attr);

                // adjust cursor location

                cu_move(row, --column);
            }
            break;
        }

// process key press from now on
do
{
    // adjust case and write

    if(opt == NAME) {
        *response= toupper(*response);
        scrn_write(row, start, dlen, response, attr);
    }

    // stop and wait for key press

    key= kb_read();

    // process key press

    switch(key)
    {
        case F1:                // return from vedit
        case F2:                // and report key press
        case F3:
        case F4:
        case F5:
        case F6:
        case F7:
        case F8:
        case F9:
        case F10:
        case UP_ARROW:
        case DOWN_ARROW:
        case UP_ARROW_K:
        case DOWN_ARROW_K:
```

```

case TAB:
case ENTER:
    cu_set_shape(cur);
    cu_rest_loc();
    return key;

case ESCAPE:
    cu_set_shape(cur);
    cu_rest_loc();
    for(i= 0; i < dlen; i++) { // restore original
        response[i]= buf[i]; // buffer contents
    }
    return key;

case CNTL_G:
case DELETE: // delete char and adjust buffer
    for(i= 0; i < stop - column + 1; i++) {
        *(rptr + i)= *(rptr + 1 + i);
    }
    *(rptr - 1 + i)= 0;
    scrn_write(row, start, dlen, response, attr);
    break;

case CNTL_T: // erase from cursor to
    while(*rptr && *rptr != ' ') // EOL
        for(i= 0; i < stop - column + 1; i++)
            *(rptr + i)= *(rptr + 1 + i);
    while(*rptr && *rptr == ' ')
        for(i= 0; i < stop - column + 1; i++)
            *(rptr + i)= *(rptr + 1 + i);
    scrn_write(row, start, dlen, response, attr);
    break;

case CNTL_END: // erase from cursor to entry end
    memset(rptr, 0, stop - column);
    scrn_write(row, start, dlen, response, attr);
    break;

case LEFT_ARROW: // move cursor left
case LEFT_ARROW_K:
    if(start_column < column) {
        cu_move(row, --column);
        rptr--;
    }
    break;

case CNTL_LEFTA: // move by word left
    if(rptr == response) {
        break;
    }
    while(*--rptr == ' ' && (int)(rptr - response) > 0);
    while(*--rptr != ' ' && (int)(rptr - response) > 0);
    if((int)(rptr - response) > 0) {
        rptr++;
    }
    cu_move(row, column= start + (int)(rptr - response));
    break;

case CNTL_RIGHTA: // move by word right
    if(*rptr) {
        while (*++rptr != ' ' && *rptr);
    }

```

14-2 Continued.

```

    }
    if(*rptr) {
        while (*++rptr == ' ' && *rptr);
    }
    cu_move(row, column= start + (int)(rptr - response));
    break;

case RIGHT_ARROW: // move cursor right
case RIGHT_ARROW_K:
    if (*rptr) {
        cu_move(row, ++column);
        rptr++;
    }
    break;

case CNTL_BS: // erase entry and start over
    memset(response, 0, dlen + 1);
    rptr= response;
    scrn_write(row, start, dlen, response, attr);
    column= start;
    cu_move(row, column);
    break;

case HOME: // go to beginning of entry
    cu_move(row, column= start);
    rptr= response;
    break;

case END: // go to end of entry
    cu_move(row, column= start + strlen(response));
    rptr= response + strlen(response);
    break;

case CNTL_H:
case BS: // move cursor back and delete
    if(column>start) {
        rptr--;
        for (i= 0; i < stop - column + 1; i++) {
            *(rptr + i)= *(rptr + 1 + i);
        }
        scrn_write(row, start, dlen, response, attr);
        cu_move(row, --column);
    }
    else {
        bleep();
    }
    break;

case INSERT: // toggle insert and overlay mode
    if(ins) // cursor size adjusted
    {
        ins=0;
        cu_rest_size();
    }
    else {
        ins= 1;
        cu_save_size();
        cu_set_size(0, 7);
    }
    break;

```



```

default:

    // mask 8 bits

    key&= 0x00ff;

    // process option

    switch (opt)
    {
        case NAME:
        case LOWER:

            // force lower case

            key= tolower(key);
            break;

        case UPPER:

            // force upper case

            key= toupper(key);
            break;

    }

    key&= 0x00ff;

    // is key printable character?

    if((key >= 0x20) && (key <= 0x7d)) {
        // is insert key toggled on

        if(ins) {
            // determine length of response

            tlen= strlen(response);

            // is response less than max response?

            if(tlen < dlen) {
                // relocate string making
                // space for new key

                for (i= dlen; i > (rptr-response); i--)
                    response[i]= response[i - 1];

                // write response buffer

                scrn_write(row, start, dlen, response, attr);

            }

        }

        // is end of edit field not reached?

        if(column < stop) {
            // write char to screen

            scrn_char(row, column, key, attr);

```

14-2 Continued.

```
        // place key in buffer
        *rptr++= (char)key;
        // adjust column 1 right
        column++;
        // move the cursor on the screen
        cu_move(row, column);
    }
    else
        // bleep at field end
        bleep();
    }
} while(1);
}
```

14-2 Ends.

15

DOS cursor management functions

This chapter presents the source code to the DOS cursor management object module that has been placed in the DOSTEXT.LIB library file. Table 15-1 repeats the cursor function prototypes.

**Table 15-1 Cursor
function prototypes.**

void	cu_get_loc(int *, int *);
int	cu_get_shape(void);
void	cu_move(int, int);
void	cu_relative_move(int, int);
void	cu_remove(void);
void	cu_display(void);
void	cu_save_size(void);
void	cu_rest_size(void);
void	cu_set_size(int, int);
void	cu_save_loc(void);
void	cu_set_shape(int);
void	cu_rest_loc(void);

Figure 15-1 presents the DOS based source code listing to CURSOR.C. This source file uses BIOS functions as building blocks for the migration library's higher level cursor management routines.

15-1 The source code listing to CURSOR.C.

```
////////////////////////////////////////
//
// cursor.c
//
// DOS version
//

#include <dos.h>

#include "tproto.h"

static UCHAR c_loc;
static UCHAR r_loc;
static UCHAR c_start;
static UCHAR c_end;

////////////////////////////////////////
//
// cu_move
//
// Moves the text cursor
//

void cu_move( int row, int col)
{
    union REGS ir, or;

        // move cursor via BIOS

        ir.h.ah= 0x02;
        ir.h.bh= 0x00;
        ir.h.dh= (UCHAR)row;
        ir.h.dl= (UCHAR)col;
        int86(0x10, &ir, &or);

}

////////////////////////////////////////
//
// cu_save_loc
//
// Saves the current cursor location
//

void cu_save_loc()
{
    union REGS ir, or;

        // save cursor location via BIOS

        ir.h.ah= 0x03;
        ir.h.bh= 0x00;
        int86(0x10, &ir, &or);
        c_loc= or.h.dl;
        r_loc= or.h.dh;
}
```

```

}

////////////////////////////////////
//
// cu_rest_loc
//
// Restores the cursor location
// which has been previously saved
//

void cu_rest_loc()
{
    // restore cursor location

    cu_move(r_loc, c_loc);
}

////////////////////////////////////
//
// cu_get_loc
//
// Gets the current location of the
// text cursor
//

void cu_get_loc(int *row, int *col)
{
    union REGS ir, or;

    // save cursor location via BIOS

    ir.h.ah= 0x03;
    ir.h.bh= 0x00;
    int86(0x10, &ir, &or);
    *col= (int)or.h.dl;
    *row= (int)or.h.dh;
}

////////////////////////////////////
//
// cu_remove
//
// Removes the text cursor from the
// screen
//

void cu_remove()
{
    union REGS ir, or;

    // remove the cursor via the BIOS

    ir.h.ah= 0x03;
    ir.h.bh= 0;

    int86(0x10, &ir, &or);

    ir.h.cl= or.h.cl;
    ir.h.ch= or.h.ch | 0x20;

```

15-1 Continued.

```
    ir.h.ah= 0x01;
    int86(0x10, &ir, &or);

}

////////////////////////////////////
//
// cu_display
//
// Displays the text cursor on the
// screen
//

void cu_display()
{
    union REGS ir, or;

    // display the cursor via the BIOS

    ir.h.ah= 0x03;
    ir.h.bh= 0;

    int86(0x10, &ir, &or);

    ir.h.ch= or.h.ch & 0xdf;
    ir.h.cl= or.h.cl;
    ir.h.ah= 0x01;
    int86(0x10, &ir, &or);
}

////////////////////////////////////
//
// cu_set_size
//
// Sets the size of the text cursor
//

void cu_set_size( int start, int end )
{
    union REGS ir, or;

    // set cursor size via the BIOS

    ir.h.ah= 0x01;
    ir.h.bh= 0;
    ir.h.ch= (UCHAR)start;
    ir.h.cl= (UCHAR)end;

    int86(0x10, &ir, &or);
}

////////////////////////////////////
//
// cu_save_size
//
```

```

// Saves the current text cursor size
//

void cu_save_size()
{
union REGS ir, or;

    // save cursor size to static variables

    ir.h.ah= 0x03;
    ir.h.bh= 0x00;

    int86(0x10, &ir, &or);

    c_start= or.h.ch;
    c_end= or.h.cl;

}

////////////////////////////////////
//
// cu_rest_size
//
// Restores the previously saved text
// cursor size
//

void cu_rest_size()
{
    cu_set_size(c_start, c_end);
}

////////////////////////////////////
//
// cu_get_shape
//
// Gets the cursor shape as an int
//

int cu_get_shape()
{
union REGS ir, or;

    // get cursor shape via the BIOS

    ir.h.ah= 0x03;
    ir.h.bh= 0x00;

    int86(0x10, &ir, &or);

    return(or.x.cx);
}

////////////////////////////////////
//
// cu_set_shape
//
// Sets the cursor shape. The shape
// is received as an int
//

```

15-1 Continued.

```
void cu_set_shape(int shape)
{
    union REGS ir, or;

    // reset the cursor shape via the BIOS
    ir.h.ah= 0x01;
    ir.x.cx= shape;
    int86(0x10, &ir, &or);
}

////////////////////////////////////
//
// bleep
//
// void function
//

void bleep()
{

}
```

15-1 Ends.

16

DOS screen management functions

This chapter presents the source code to the DOS screen management object, the RECT.OBJ object module and the MAKE.OBJ object module that have been placed in the DOSTEXT.LIB library file. Table 16-1 repeats the screen function prototypes.

Table 16-1 Screen function prototypes.

void	scrn_write(int row, int col, int len, char *str, UCHAR attr);
void	scrn_init(void);
int	scrn_read_char(unsigned short, unsigned short);
void	scrn_save(void);
void	scrn_restore(void);
void	scrn_clear(void);
void	scrn_attr(int, int, int, unsigned char);
void	scrn_change_attr(unsigned char);
void	scrn_char(int row, int col, char ch, UCHAR attr);
void	scrn_repeat_char(int row, int col, int len, char ch, UCHAR attr);
void	scrn_chr(int, int, int);

There are three source modules that all contribute to the library's screen management functionality. These three source files are MAKE.C

(FIG. 16-1), RECT.C (FIG. 16-2), and SCREEN.C (FIG. 16-3).

Figure 16-1 presents the source code listing to MAKE.C. Functions present in this source module facilitate the handling of screen attributes and tokens.

16-1 The source code listing to MAKE.C.

```
////////////////////////////////////
//
// make.c
//
// DOS version
//

//
// include file
//

#include "tproto.h"

////////////////////////////////////
//
// mk_attr
//
// Makes an attribute
//

UCHAR mk_attr(UCHAR fore,
              UCHAR back,
              UCHAR inten,
              UCHAR blink)
{
    UCHAR attr;
    attr= back << 4;
    return((UCHAR) attr | fore | inten | blink );
}

////////////////////////////////////
//
// mk_attr_intense
//
// Makes an existing attribute
// intense
//

UCHAR mk_attr_intense(UCHAR attr)
{
    return(attr | ON_INTENSITY);
}

////////////////////////////////////
//
// mk_attr_intense_blink
//
// Makes an existing attribute intense
// and blink
//
```

```

    UCHAR mk_attr_intense_blink(UCHAR attr)
    {
        return(attr | ON_INTENSITY | ON_BLINK);
    }

```

```

    //////////////////////////////////////
    //
    // mk_attr_blink
    //
    // Makes an existing attribute blink
    //

```

```

    UCHAR mk_attr_blink(UCHAR attr)
    {
        return(attr | ON_BLINK);
    }

```

```

    //////////////////////////////////////
    //
    // mk_attr_inverse
    //
    // Makes an existing attribute
    // inverse
    //

```

```

    UCHAR mk_attr_inverse(UCHAR attr)
    {
        UCHAR u1,u2;
        u1= attr & 0xff;
        u2= u1;
        u2<<= 4;
        return (u2 | (u1 >> 4)) & 0x7f;
    }

```

```

    //////////////////////////////////////
    //
    // mk_token
    //
    // Makes a token
    //

```

```

    int mk_token(UCHAR ch, UCHAR attr)
    {
        int ret_val;
        ret_val= 0;
        ret_val= ch;
        return ((ret_val << 8) | attr);
    }

```

```

    //////////////////////////////////////
    //
    // mk_char_attr
    //
    // Takes a screen token and returns an
    // 8-bit char and 8-bit unsigned char
    // attribute via pointers passed as
    // parameters

```

16-1 Continued.

```
//

void mk_char_attr(int token, char *ch, UCHAR *attr)
{
    *attr= (UCHAR)token & 0x00ff;
    token>>= 8;
    *ch= (UCHAR) token;
}
```

16-1 Ends.

Figure 16-2 presents the source code listing to RECT.C. This source module presents functions that facilitate the management of rectangular portions of the screen.

16-2 The source code listing to RECT.C.

```
////////////////////////////////////
//
// rect.c
//
// DOS Version
//

#include <stdlib.h>
#include <stdio.h>
#include <malloc.h>
#include <string.h>

#include "tproto.h"

static
char buff32[80] = { 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32 };

////////////////////////////////////
//
// saveRect
//
// Saves a rectangular image to
// memory
//

void saveRect(RECT *R)
{
    unsigned int *iptr;
    int row;
    int column;
```

```

        iptr= (unsigned int *)R->image;
        for(row= R->ul_row; row <= R->lr_row; row++) {
            for(column= R->ul_col; column <= R->lr_col; column++) {
                *iptr++= scrn_read_char(row, column);
            }
        }
    }
}

```

```

////////////////////////////////////
//
// restRect
//
// Restores a previously saved
// rectangular image
//

```

```

void restRect(RECT *R)
{
    unsigned int *iptr;
    int          row, token;
    int          column;
    UCHAR        attr;
    char         ch;

    iptr= (unsigned int *) R->image;

    for(row= R->ul_row; row <= R->lr_row; row++) {
        for(column= R->ul_col; column <= R->lr_col; column++) {
            token= *iptr++;
            mk_char_attr(token, &ch, &attr);
            scrn_char(row, column, ch, attr);
        }
    }
}

```

```

////////////////////////////////////
//
// setRect
//
// Initializes a RECT structure
//

```

```

RECT *setRect(RECT *R, int ur, int uc, int lr, int lc)
{
    int size;
    R= (RECT *)malloc(sizeof(RECT));
    if(!R) {
        scrn_clear();
        cu_display();
        cu_move(0, 0);
        printf("Memory allocation Error: setRect\n");
        exit(0);
    }
    R->ul_row= ur;
    R->ul_col= uc;
    R->lr_row= lr;
    R->lr_col= lc;
    size= sizeRect(R);
}

```

16-2 Continued.

```
    R->image= (unsigned int *)calloc(size, sizeof(int));
    return(R);
}
```

```
////////////////////////////////////
//
// sizeRect
//
// Calculates the size of a rectangular
// screen image
//
```

```
unsigned int sizeRect(RECT *R)
{
    int height, width, size;

    height= R->lr_row - R->ul_row + 1;
    width= R->lr_col - R->ul_col + 1;
    size= height * width;
    return(size);
}
```

```
////////////////////////////////////
//
// dsyRect
//
// Destroys a rectangular structure
// by freeing previously allocated
// memory
//
```

```
void dsyRect(RECT *R)
{
    free((void *)R->image);
    free((void *)R);
}
```

```
////////////////////////////////////
//
// subRect
//
// Subtracts a one RECT structure
// from another
//
```

```
void subRect( RECT *destR, RECT *srceR )
{
    destR->ul_row-= srceR->ul_row;
    destR->ul_col-= srceR->ul_col;
    destR->lr_row-= srceR->lr_row;
    destR->lr_col-= srceR->lr_col;
}
```

```
////////////////////////////////////
//
// addRect
//
```

```

// Adds one RECT structure to another
//

void addRect(RECT *destR, RECT *srceR)
{
    destR->ul_row+= srceR->ul_row;
    destR->ul_col+= srceR->ul_col;
    destR->lr_row+= srceR->lr_row;
    destR->lr_col+= srceR->lr_col;
}

////////////////////////////////////
//
// dupRect
//
// Duplicates one RECT structure
// with another

void dupRect( RECT *destR, RECT *srceR)
{
    destR->ul_row= srceR->ul_row;
    destR->ul_col= srceR->ul_col;
    destR->lr_row= srceR->lr_row;
    destR->lr_col= srceR->lr_col;
}

////////////////////////////////////
//
// clrRect
//
// Clears a RECT structure using the
// NORMAL (7) screen attribute
//

void clrRect(RECT *R)
{
    int row;
    int column;
    int row_stop, col_stop;
    int token;

    row_stop= R->lr_row;
    col_stop= R->lr_col;

    for(row= R->ul_row; row < row_stop; row++) {
        for(column= R->ul_col; column < col_stop; column++) {
            scrn_char( row, column, ' ', 7 );
        }
    }
}

////////////////////////////////////
//
// offRect
//
// Shifts the row and column locations
// within a RECT structure by the
// offset values.

```

16-2 Continued.

```
//  
  
void offRect( RECT *destR, int row_off, int col_off )  
{  
    destR->ul_row+= row_off;  
    destR->ul_col+= col_off;  
    destR->lr_row+= row_off;  
    destR->lr_col+= col_off;  
}  
  
////////////////////////////////////  
//  
// boxRect  
//  
// Places a border around a rectangle  
//  
  
void boxRect(RECT *R, int box_type, unsigned char attr)  
{  
    int row,column;  
    int token;  
    int top_bot, left_right, ul, ur, ll, lr; // box choices  
  
    switch(box_type)  
    {  
        case 1:  
            top_bot= 196;  
            left_right= 186;  
            ul= 214;  
            ur= 183;  
            ll= 211;  
            lr= 189;  
            break;  
        case 2:  
            top_bot= 205;  
            left_right= 179;  
            ul= 213;  
            ur= 184;  
            ll= 212;  
            lr= 190;  
            break;  
        case 3:  
            top_bot= 205;  
            left_right= 186;  
            ul= 201;  
            ur= 187;  
            ll= 200;  
            lr= 188;  
            break;  
        default:  
            top_bot= 196;  
            left_right= 179;  
            ul= 218;  
            ur= 191;  
            ll= 192;  
            lr= 217;  
            break;  
    }  
  
    for(row= R->ul_row; row < R->lr_row; row++) {
```



```

        scrn_write(row, R->ul_col, R->lr_col - R->ul_col, buff32, attr);
    }

    // draw top and bottom
    for(column= R->ul_col; column < R->lr_col; ++column) {
        scrn_char(R->ul_row, column, top_bot, attr);
        scrn_char(R->lr_row, column, top_bot, attr);
    }

    // draw left and right borders
    for(row= R->ul_row; row < R->lr_row; ++row) {
        scrn_char(row, R->ul_col, left_right, attr);
        scrn_char(row, R->lr_col, left_right, attr);
    }

    // plop the four corners
    scrn_char(R->ul_row, R->ul_col, ul, attr);
    scrn_char(R->ul_row, R->lr_col, ur, attr);
    scrn_char(R->lr_row, R->ul_col, ll, attr);
    scrn_char(R->lr_row, R->lr_col, lr, attr);
}

```

16-2 Ends.

Figure 16-3 presents the DOS-based source code listing to SCREEN.C. This source file uses direct video memory access techniques as a route to implementing the migration library's higher-level screen management routines. Directly accessing screen memory produces very professional results.

16-3 The source code listing to SCREEN.C.

```

////////////////////////////////////////
//
// screen.c
//
// DOS version
//
//
// include files here
//
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <dos.h>

#include "tproto.h"

//
// static data
//

```

16-3 Continued.

```
static
char buff32[80] = { 32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32,
                    32, 32, 32, 32, 32, 32, 32, 32, 32, 32 };
```

```
//
// global variables
//

UCHAR far    *ScrnPtr;
static UINT  scrn_buf[80 * 25];

////////////////////////////////////
//
// scrn_init
//
// Initialize the ScrnPtr variable
//

void scrn_init()
{
union REGS ir, or;

    // determine if monitor is color
    // or mono and set ScrnPtr to
    // appropriate value

    ir.h.ah= 0x0f;

    // via the BIOS

    int86(0x10, &ir, &or);

    // if monitor type is Mono

    if(or.h.al == 7) {
        ScrnPtr= (UCHAR far *)0x0b0000000L;
    }

    // else is Color

    else {
        ScrnPtr= (UCHAR far *)0x0b8000000L;
    }

}

////////////////////////////////////
//
// scrn_char
//
// Write a character to the screen
```

```

//
void scrn_char(int row, int col, char ch, UCHAR attr)
{
    int      offset;
    UCHAR far *p;

    // set p to upper left hand corner of text screen
    p= (UCHAR far *)ScrnPtr;

    // calculate screen offset for string write start
    offset= (row * 160) + (col * 2);

    // add offset to p
    p+= offset;

    // move the character to the screen
    *p++= ch;

    // move the attribute to the screen
    *p++= attr;
}

////////////////////////////////////
//
// scrn_repeat_char
//
// Repeats a character on a horizontal
// line

void scrn_repeat_char(int row, int col, int len, char ch, UCHAR attr)
{
    int counter;

    for(counter= 0; counter < len; counter++) {
        scrn_char(row, col++, ch, attr);
    }
}

////////////////////////////////////
//
// scrn_attr
//
// Change a specified number of screen
// attributes without altering the
// displayed characters
//

void scrn_attr(int row, int col, int len, UCHAR attr)
{
    int      offset, counter;
    UCHAR far *p;

    // set p to upper left hand corner of text screen

```

16-3 Continued.

```

    p= (UCHAR far *)ScrnPtr;

    // calculate screen offset for string write start
    offset= (row * 160) + (col * 2);

    // add offset to p
    p+= offset;

    // Change the attributes
    for(counter= 0; counter < len; counter++) {

        // bypass the character

        p++;

        // move the attribute to the screen
        *p+= attr;

    }

}

////////////////////////////////////
//
// scrn_write
//
// Write a string to the screen
//

void scrn_write(int row, int col, int len, char *str, UCHAR attr)
{
    int      offset, counter;
    UCHAR far *p;

    // set p to upper left hand corner of text screen
    p= (UCHAR far *)ScrnPtr;

    // calculate screen offset for string write start
    offset= (row * 160) + (col * 2);

    // add offset to p
    p+= offset;

    // if length is 0 then calculate the length
    // of the string
    if(!len) {
        len= strlen(str);
    }

    // write the string to the screen

```

```

    for(counter= 0; counter < len; counter++) {

        // move the character to the screen

        *p++= *str++;

        // move the attribute to the screen

        *p++= attr;

    }

}

////////////////////////////////////
//
// scrn_clear
//
// Clear the screen with the NORMAL
// (7) attribute
//

void scrn_clear()
{
    int row;

    for(row= 0; row < 25; row ++){
        scrn_write(row, 0, 80, buff32, 7);
    }
}

////////////////////////////////////
//
// scrn_change_attr
//
// Change the attribute of the screen
// without altering the displayed
// characters
//

void scrn_change_attr(unsigned char attr)
{
    int row;

    for(row= 0; row < 25; row ++){
        scrn_attr(row, 0, 80, attr);
    }
}

////////////////////////////////////
//
// scrn_chr
//
// Write a 16-bit token to the screen.
//

void scrn_chr(int row, int col, int token)
{
    UCHAR at;
    char ch;

```

16-3 Continued.

```
    ch= (char)token;
    token = token >> 8;
    at= (UCHAR)token;

    scrn_char(row, col, ch, at);
}

////////////////////////////////////
//
// scrn_read_char
//
// Read the character and its attribute
// at a specified screen location
//

int scrn_read_char(USHORT row, USHORT col)
{
    int      offset, token;
    UCHAR    buffer[2];
    UCHAR far *p;

    // set p to upper left hand corner of text screen
    p= (UCHAR far *)ScrnPtr;

    // calculate screen offset for string write start
    offset= (row * 160) + (col * 2);

    // add offset to p
    p+= offset;

    // move the character to the buffer
    buffer[0]= *p++;

    // move the attribute to the buffer
    buffer[1]= *p;

    token= mk_token(buffer[0], buffer[1]);

    // and return
    return token;
}

////////////////////////////////////

void wrBox(WIND *W)
{
    int row, column;
    int top_bot, left_right, ul, ur, ll, lr; // box choices

    switch(W->box_type)
    {
        case 1:
            top_bot = 196;
            left_right = 186;
```

```

        ul = 214;
        ur = 183;
        ll = 211;
        lr = 189;
        break;
    case 2:
        top_bot = 205;
        left_right = 179;
        ul = 213;
        ur = 184;
        ll = 212;
        lr = 190;
        break;
    case 3:
        top_bot = 205;
        left_right = 186;
        ul = 201;
        ur = 187;
        ll = 200;
        lr = 188;
        break;
    default:
        top_bot = 196;
        left_right = 179;
        ul = 218;
        ur = 191;
        ll = 192;
        lr = 217;
        break;
}

for(row=W->ul_row; row<W->lr_row; row++) {
    scrn_write(row, W->ul_col, W->lr_col - W->ul_col, buff32, W->attr);
}

// draw top and bottom
for(column=W->ul_col; column<W->lr_col; ++column) {
    scrn_char(W->ul_row, column, top_bot, W->attr);
    scrn_char(W->lr_row, column, top_bot, W->attr);
}

// draw left and right borders
for(row=W->ul_row; row<W->lr_row; ++row) {
    scrn_char(row, W->ul_col, left_right, W->attr);
    scrn_char(row, W->lr_col, left_right, W->attr);
}

// plop the four corners
scrn_char(W->ul_row, W->ul_col, ul, W->attr);
scrn_char(W->ul_row, W->lr_col, ur, W->attr);
scrn_char(W->lr_row, W->ul_col, ll, W->attr);
scrn_char(W->lr_row, W->lr_col, lr, W->attr);
}

```

16-3 Continued.

```
unsigned int sizeImg(WIND *W)
{
    int height, width, size;
    height= W->lr_row - W->ul_row;
    width= W->lr_col-W->ul_col;
    ++height;
    ++width;
    size= height * width;
    return(size);
}

void vdBox(RECT *R, int box_type, UCHAR attr)
{
    int row,column;
    int top_bot, left_right, ul, ur, ll, lr; // box choices

    switch(box_type)
    {
        case 1:
            top_bot = 196;
            left_right = 186;
            ul = 214;
            ur = 183;
            ll = 211;
            lr = 189;
            break;
        case 2:
            top_bot = 205;
            left_right = 179;
            ul = 213;
            ur = 184;
            ll = 212;
            lr = 190;
            break;
        case 3:
            top_bot = 205;
            left_right = 186;
            ul = 201;
            ur = 187;
            ll = 200;
            lr = 188;
            break;
        default:
            top_bot = 196;
            left_right = 179;
            ul = 218;
            ur = 191;
            ll = 192;
            lr = 217;
            break;
    }

    for(row=R->ul_row; row<R->lr_row; row++) {
        scrn_write(row, R->ul_col, R->lr_col - R->ul_col, buff32, attr);
    }

    // draw top and bottom
    for(column=R->ul_col; column<R->lr_col; ++column) {
        scrn_char(R->ul_row, column, top_bot, attr);
        scrn_char(R->lr_row, column, top_bot, attr);
    }
}
```



```

    }

    // draw left and right borders
    for(row=R->ul_row; row<R->lr_row; ++row) {
        scrn_char(row, R->ul_col, left_right, attr);
        scrn_char(row, R->lr_col, left_right, attr);
    }

    // plop the four corners
    scrn_char(R->ul_row, R->ul_col, ul, attr);
    scrn_char(R->ul_row, R->lr_col, ur, attr);
    scrn_char(R->lr_row, R->ul_col, ll, attr);
    scrn_char(R->lr_row, R->lr_col, lr, attr);
}

////////////////////////////////////
//
// scrn_save
//
// Saves the screen to a static buffer
//

void scrn_save()
{
    int row, col;
    UINT *S;

    // set S to screen buffer

    S= scrn_buf;

    // save screen by row
    for(row= 0; row < 25; row++) {
        // save by column
        for(col= 0; col < 80; col++) {
            // save screen token
            *S= scrn_read_char(row, col);

            S++;
        }
    }
}

////////////////////////////////////
//
// scrn_restore
//
// restores the previously saved
// screen

```

16-3 Continued.

```

//

void scrn_restore()
{
    UINT      *SB;
    UCHAR      attr;
    int        row, col, token;
    char        ch;

    // set SB to screen buffer

    SB= scrn_buf;

    // restore by row

    for(row= 0; row < 25; row++) {

        // restore by column

        for(col= 0; col < 80; col++) {

            // split token to char and attr

            mk_char_attr(*SB++, &ch, &attr);

            // write the char and attr to the screen

            scrn_char(row, col, ch, attr);

        }

    }

}

////////////////////////////////////
//
// rdImg
//
// Transfers a rectangular region of the screen
// to buffer and blanks the area
//

void rdImg(WIND *W)
{
    int    row, column;
    UINT   *buf_ptr;

    buf_ptr = (unsigned int *)W->img_ptr;

    for(row=W->ul_row; row<=W->lr_row; row++) {

        for( column=W->ul_col; column<=W->lr_col; column++) {
            *buf_ptr= scrn_read_char(row, column);
            buf_ptr++;
        }

    }

}

////////////////////////////////////
//

```

```

// wrImg
//
// transfers a rectangular region of the screen
// to buffer and blanks the area
//

void wrImg(WIND *W)
{
    int    row, column, token;
    UINT   *img_ptr;
    UCHAR  attr;
    char   ch;

    img_ptr = (UINT *)W->img_ptr;

    for(row=W->ul_row; row<=W->lr_row; row++) {
        for( column=W->ul_col; column<=W->lr_col; column++) {
            // split token to char and attr
            mk_char_attr(*img_ptr, &ch, &attr);

            img_ptr++;

            scrn_char(row, column, ch, attr);
        }
    }

    //////////////////////////////////////
    //
    // rdWind
    //
    // transfers a rectangular region of the screen
    // to buffer and blanks the area
    //
    //

    void rdWind(WIND *W)
    {
        int row, column;
        UINT *buf_ptr;

        buf_ptr = (UINT *)W->wind_ptr;

        for(row=W->ul_row; row<=W->lr_row; row++) {
            for( column=W->ul_col; column<=W->lr_col; column++) {
                *buf_ptr= scrn_read_char(row, column);

                buf_ptr++;
            }
        }

        //////////////////////////////////////
        //
        // wrWind
        //

```

16-3 Continued.

```
// transfers a rectangular region of the screen
// to buffer and blanks the area
//
//

void wrWind(WIND *W)
{
    UCHAR attr;
    UINT  *img_ptr;
    char  ch;
    int   row, column, token;

    // set the ipointer to window image

    img_ptr = (UINT *)W->wind_ptr;

    // write window by row

    for(row=W->ul_row; row<=W->lr_row; row++) {

        // write window by column

        for( column=W->ul_col; column<=W->lr_col; column++) {

            // split token to char and attr

            mk_char_attr(*img_ptr, &ch, &attr);
            img_ptr++;

            scrn_char(row,column, ch, attr);
        }
    }
}
```

16-3 Ends.

17

DOS mouse management functions

This chapter presents the source code to the DOS mouse management object module that has been placed in the DOSTEXT.LIB library file. Table 17-1 repeats the mouse function prototypes.

Table 17-1 High level mouse function prototypes.

int	ms_init(void);
void	ms_on(void);
void	ms_off(void);
int	ms_status(int *x, int *y);
void	ms_map_display(int row, int col, int key_val);

Figure 17-1 presents the DOS based source code listing to MOUSE.C. This function uses BIOS functions for the implementation of mouse management functions in the DOSTEXT.LIB library file.

17-1 The source code listing to MOUSE.C.

```
////////////////////////////////////////  
//  
// mouse.c  
//  
// DOS version  
//
```

17-1 Continued.

```
#include <dos.h>
#include <memory.h>
#include <stdio.h>
#include "tproto.h"

////////////////////////////////////
//
// ms_init
//
// Initialize the mouse

int ms_init()
{
    union REGS ir, or;
    int button;

    ir.x.ax= 0x00;
    int86(0x33, &ir, &or);
    if(!or.x.ax) {
        button= 0xffff;
    }
    else {
        button= or.x.bx;
    }

    return button;
}

////////////////////////////////////
//
// ms_off
//
// Turns the mouse off
//

void ms_off()
{
    union REGS ir, or;

    ir.x.ax= 0x02;
    int86(0x33, &ir, &or);
}

////////////////////////////////////
//
// ms_on
//
// Turns the mouse on
//

void ms_on()
{
    union REGS ir, or;

    ir.x.ax= 0x01;
    int86(0x33, &ir, &or);
}
```

```

////////////////////////////////////
//
// ms_status
//
// Returns the mouse button status and
// the row and column location of the
// mouse
//

int ms_status(int *x, int *y)
{
union REGS ir, or;

    ir.x.ax= 0x03;
    int86(0x33, &ir, &or);
    *x= or.x.cx;
    *y= or.x.dx;

    return or.x.bx;
}

////////////////////////////////////
//
// ms_map_display
//
// This function reports the mouse
// location on a button press. It
// will prove invaluable when mapping
// a user interface
//

void ms_map_display(int row, int col, int key_val)
{
    UCHAR  exit_flag= 0;
    char   buffer[20];
    int    key;
    int    x= 0, y= 0;

    do {

        key= kb_status();

        if(!key) {
            key= ms_status(&x, &y);
            if(key == 1) {
                scrn_write(row, col, 0, "Left Button Press ", 7);
            }
            if(key == 2) {
                scrn_write(row, col, 0, "Right Button Press", 7);
            }
            memset(buffer, 0, 20);
            sprintf(buffer, "x= %03d  y= %03d", x, y);
            scrn_write(row + 1, col, 0, buffer, 7);
        }

        if(key == key_val) {
            exit_flag= 1;
        }

    } while(!exit_flag);
}

```

17-1 Ends.

18

DOS window management functions

This chapter presents the source code to the DOS window management object module that has been placed in the DOSTEXT.LIB library file. Table 18-1 repeats the high level window function prototypes.

Table 18-1 High level window function prototypes.

WIND	*wind_init(WIND *W_PTR, int ulr, int ulc, int lrr, int lrc, UCHAR attr, int border, char *title);
int	wind_kb_edit(WIND *W, char *response, int row, int column, int dlen, int opt, UCHAR attr);
void	wind_display(WIND *);
void	wind_remove(WIND *);
void	wind_attr(WIND *, int, int, int, unsigned char);
void	wind_write(WIND *, int, int , int , char *, UCHAR);

Table 18-1 Continued.

void	wind_char(WIND *, int row, int col, char ch, UCHAR attr);
void	wind_repeat_char(WIND *, int row, int col, int len, char ch, UCHAR attr);
void	wind_destroy(WIND *);
int	wind_read_char(WIND *, int, int);
void	wind_clear(void);
void	wind_cu_move(WIND *, int, int);

Figure 18-1 presents the DOS-based source code listing to WINDOW.C. This source file uses functions presented in RECT.C, MAKE.C, and SCREEN.C as building blocks for the migration library's higher level window management routines.

18-1 The source code listing to WINDOW.C.

```
////////////////////////////////////
//
// window.c
//
// DOS version
//

#include <stdlib.h>
#include <malloc.h>
#include <string.h>
#include <stdio.h>

#include "tproto.h"

#define wNULL 0
#define W_SIZE sizeof(WIND)
extern unsigned int sizeImg(WIND *);

////////////////////////////////////
//
// wind_destroy
//
// Destroys a window structure by
// freeing previously dynamically
// allocated memory
//

void wind_destroy(WIND *W)
{
    if(W->img_ptr != wNULL) {
        free((void *)W->img_ptr);
    }

    if(W->wind_ptr != wNULL) {
        free((void *)W->wind_ptr);
    }
}
```

```

        if(W->t_title != wNULL) {
            free((void *)W->t_title);
        }

        if(W != wNULL) {
            free((void *)W);
        }
    }

    //////////////////////////////////////
    //
    // strt_wind
    //
    // This function is called the first
    // time that a window is displayed
    //

void strtWind(WIND *W)
{
    char *tptr;

    tptr= W->t_title;

    rdImg(W);

    wrBox(W);

    rdWind(W);

    W->visible= 1;

    if(W->show_top) {
        wind_write(W, 0, W->top_offset, W->top_length, tptr, W->attr);
    }
}

    //////////////////////////////////////
    //
    // setWind
    //
    // Preps the window structure
    //

WIND *setWind(WIND *W, int ul_row, int ul_col, int lr_row, int lr_col)
{
    W= (WIND *)calloc(W_SIZE,sizeof(char));
    if(!W) {
        scrn_clear();
        cu_display();
        cu_move(0, 0);
        printf("Memory allocation Error: setWind\n");
        exit(0);
    }

    W->ul_row= ul_row;
    W->ul_col= ul_col;

```

18-1 Continued.

```
W->lr_row= lr_row;
W->lr_col= lr_col;
W->img_size= sizeImg(W);
W->img_ptr= (unsigned int *)calloc(W->img_size, sizeof(int));
if(!W->img_ptr) {
    scrn_clear();
    cu_display();
    cu_move(0, 0);
    printf("Memory allocation Error: W->img_ptr\n");
    exit(0);
}

W->wind_ptr = (unsigned int *)calloc(W->img_size, sizeof(int));
if(!W->wind_ptr) {
    scrn_clear();
    cu_display();
    cu_move(0, 0);
    printf("Memory allocation Error: setWect\n");
    exit(0);
}

W->visible= aFALSE;
W->box_type =0;
W->attr= NORMAL;
W->t_title= 0;
W->b_title= 0;
W->show_top= aFALSE;
W->show_bot= aFALSE;
return(W);
}

////////////////////////////////////
//
// setBord
//
// Sets the border variable in the
// window structure
//
// set the border
//
// #   T B L W
// - - - - -
// 0 = S_S_S_S
// 1 = S_S_D_D
// 2 = D_D_S_S
// 3 = D_D_D_D
//
//

void setBord(WIND *W, int type)
{
    W->box_type= type;
}

////////////////////////////////////
//
// wind_attr
//
```

```

// Sets a series of attributes on a row
// in a window
//

void wind_attr(WIND *W, int row, int col, int length, UCHAR attr)
{
    row+= W->ul_row;
    col+= W->ul_col;
    scrn_attr( row, col, length, attr);
}

////////////////////////////////////
//
// wind_remove
//
// Removes a window from the screen
// by restoring the screen which
// had previously been under the window
// area
//

void wind_remove(WIND *W)
{
    if(W->visible) {
        rdWind(W);
        wrImg(W);
        W->visible= 0;
    }
}

////////////////////////////////////
//
// setAttr
//
// Sets the attribute in the window
// structure
//

void setAttr(WIND *W, UCHAR attr)
{
    W->attr= attr;
}

void
wind_display(W)
WIND *W;
{
    if(!W->visible)
    {
        rdImg(W);
        wrWind(W);
        W->visible = 1;
    }
}

////////////////////////////////////
//
// setTitle

```

18-1 Continued.

```
//
// Prepares the window structure
// to receive the title string and
// title location
//

void setTitle(WIND *W, char *top)
{
    W->top_length= strlen(top);
    W->top_offset= ( (W->lr_col - W->ul_col) - W->top_length )/2;
    W->top_offset+= 1;
    W->t_title= (char *)malloc(W->top_length + 1);
    if(!W->t_title) {
        scrn_clear();
        cu_display();
        cu_move(0, 0);
        printf("Memory allocation Error: setTitle\n");
        exit(0);
    }
    memset(W->t_title, '\0', W->top_length + 1);
    strcpy(W->t_title, top);
    W->show_top= aTRUE;
}

////////////////////////////////////
//
// wind_write
//
// Writes a character string to the
// window
//

void wind_write(WIND *W,
                int row,
                int col,
                int length,
                char *str,
                UCHAR attr)
{
    int count;

    if(!length) {
        length= strlen(str);
    }

    if(col==CENTER) {
        col=( W->lr_col - W->ul_col - length-1) / 2;
    }
    row+= W->ul_row;
    col+= W->ul_col;

    scrn_write( row, col, length, str, attr);
}

////////////////////////////////////
//
// wind_init
```

```

//
// Handles all of the mid-level window
// initialization functions
//

WIND *wind_init(WIND *W_PTW,
                int ulr,
                int ulc,
                int lrr,
                int lrc,
                unsigned char attr,
                int border,
                char *title )
{
    // Allocate memory and return pointer to structure

    W_PTW = setWind(W_PTW,ulr,ulc,lrr,lrc);

    // Set Window Attr - Fore,Back,Intensity,Blink

    setAttr(W_PTW, attr);

    // Set Window Border

    setBord(W_PTW,border);

    // Set the bottom title

    setTitle(W_PTW,title);

    // Display window

    strtWind(W_PTW);

    // return the pointer

    return W_PTW;
}

////////////////////////////////////////
//
// wind_kb_edit
//
// Edit an alphanumeric field in a
// window
//

int wind_kb_edit(WIND *W,
                 char *response,
                 int row,
                 int column,
                 int dlen,
                 int opt,
                 UCHAR attr)
{
    int ret_val;

    ret_val= kb_edit(response,
                     row + W->ul_row,
                     column + W->ul_col,
                     dlen,

```

18-1 Continued.

```
        opt,
        attr);
    return(ret_val);
}

////////////////////////////////////
//
// wind_repeat_char
//
// Repeats a character on a row in a
// window
//

void wind_repeat_char(WIND *W,
                      int row,
                      int col,
                      int len,
                      char ch,
                      UCHAR attr)
{
    scrn_repeat_char(W->ul_row + row,
                     W->ul_col + col,
                     len,
                     ch,
                     attr);
}

////////////////////////////////////
//
// wind_char
//
// Writes a character to a window
// location
//

void wind_char(WIND *W, int row, int col, char ch, UCHAR attr)
{
    scrn_char(W->ul_row + row,
              W->ul_col + col,
              ch,
              attr);
}
```

18-1 Ends.

19

DOS printer management functions

In DOS, you manage the printer via BIOS functions. The DOS printer functions involve sending bytes to the printer. These foundation operations have been used to create some higher level printer management functions. There are two dummy printer functions that facilitate compatibility with the OS/2 library.

This chapter presents the source code to the DOS printer management object module that has been placed in the DOSTEXT.LIB library file. Table 19-1 repeats all the printer function prototypes.

Table 19-1 Printer function prototypes.

int	print_open(int num);
int	print_close(int num);
int	print_newline(int num);
int	print_cr(int num);
int	print_string(int num, char *);
int	print_char(int, char);
int	print_scrn(int);
int	print_scrnFF(int);
int	print_status(int);
void	print_set_column(int, int);

Figure 19-1 presents the DOS-based source code listing to PRINTER.C. This function uses DOS's printer functions that reflect object modules contained in the DOSTEXT.LIB library file, FIG. 19-1. The source code listing to PRINTER.C.

19-1 The source code listing to PRINTER.C.

```

////////////////////////////////////
//
// printer.c
//
// DOS Version
//

#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <memory.h>
#include <malloc.h>

#include "tproto.h"

////////////////////////////////////
//
// print_string
//
// Send a string of bytes to the
// printer
//

int print_string(int num, char *buffer)
{
    int ret_val, counter, len;

    len= strlen(buffer);
    for(counter= 0; counter < len; counter++) {
        print_char(num, *buffer++);
    }
    return 0;
}

////////////////////////////////////
//
// print_char
//
// Send a character to the printer
//

int print_char(int num, char ch)
{
    union REGS ir, or;

    ir.x.dx= num;
    ir.h.ah= 0x00;
    ir.h.al= ch;
    int86(0x17, &ir, &or);
    return(mk_token(or.h.ah, 0));
}

```

```

}

////////////////////////////////////
//
// print_newline
//
// Send a newline character to the
// printer
//

int print_newline(int num)
{
    return((int)print_char(num, '\n'));
}

////////////////////////////////////
//
// print_cr
//
// Send a carriage return to the
// printer
//

int print_cr(int num)
{
    return((int)print_char(num, aCR));
}

////////////////////////////////////
//
// print_open
//
// Void function required for OS/2
// migration
//

int print_open(int num)
{
    num= 0;
    return num;
}

////////////////////////////////////
//
// print_close
//
// Void function required for OS/2
// migration
//

int print_close(int num)
{
    num= 0;
    return num;
}

////////////////////////////////////
//
// print_set_column
//

```

19-1 Continued.

```
// Sets the printer head to a specified
// column location. This function will
// prove useful is creating formatted
// printer output.
//

void print_set_column(int num, int column)
{
    int ctr;

    print_char(num, aCR);

    for(ctr= 0; ctr < column; ctr++) {
        print_char(num, ' ');
    }
}
```

19-1 Ends.

Epilogue

I believe OS/2 to be a very exciting operating system with an extraordinary amount of potential. This potential, in part, is based on OS/2's ability to run DOS, Windows, OS/2 Full Screen and OS/2 Presentation Manager programs.

OS/2's multitasking and multithreading capabilities are powerful features of the operating system. When I first heard about the possibility of true multitasking capabilities on the PC, I cynically reasoned that no real person would ever need to use them. Boy, was I wrong.

My reasons for selecting to do an OS/2 Full Screen title were simple enough. There are still far more programs and machines running under DOS than under Windows or OS/2. Although industry leaders feel that protected mode multitasking graphically based operating systems are the wave of the future, there are still millions of users still married to the DOS text interface. I understand why.

I've yet to find a graphical interface based word processor or text editor that is easier for my middle-aged eyes than a character mode one. Even though I've got both Windows and OS/2 Presentation Manager based word processors and text editors in my program library, I still use the character mode version of Brief (DOS session or OS/2 Full Screen session) and Word Perfect (in a DOS session launched by OS/2) all the time.

What I'm saying is that moving your DOS applications to the 32-bit flat memory model world of Full Screen OS/2 makes great sense. You'll

have tons of memory available for data, and your users won't have to move to a new generation graphical interface arena until they're ready.

It is my hope that this book has eased your path for migrating your DOS character mode program to OS/2. Please know that I'm always interested in reading your comments and seeing any library routines you've added to those presented in the book. Feel free to write me via the publisher and I will answer as time permits.

Namaste',

Len

ldorfman@delphi.com

Index

A

addRect, 304, 360-361
Application Programmer's Interface (API), xiv
 full-screen OS/2 & DOS character mode,
 1-22
 libraries/demonstration programs, 18-22

B

border styles, 106
boxRect, 52, 306, 362

C

clrRect, 305, 361
color, screen, 43
cursor management, 35-41
 DOS functions, 349-354
 function prototypes, 35, 293, 349-354
 manipulating location/size/visibility, 38-41
 OS/2 full-screen functions, 293-297
cu_display, 37, 38, 295-296, 352
cu_get_loc, 36, 295, 351
cu_get_shape, 36, 297, 353
cu_move, 36, 38, 295, 350
cu_relative_move, 36-37
cu_remove, 37, 38, 295, 351
cu_rest_loc, 38, 38, 294, 351
cu_rest_size, 37, 38, 296, 353
cu_save_loc, 37-38, 294, 350
cu_save_size, 37, 38, 296, 352-353

cu_set_shape, 36, 297, 353
cu_set_size, 37, 296, 352

D

demonstration programs (*see* source code)
dialog boxes
 alphanumeric data field entry, 119-129
 filename/file size, 141-143
 mapping, 111-119
 multiple item check, 129-138
 string list, 138-141
disk operating system (DOS)
 full-screen character mode API, 1-22
 function prefix list, 4
dsyRect, 52, 304, 360
dupRect, 305, 361

F

functions
 cursor, 35-38
 cursor (DOS), 349-354
 cursor (OS/2), 293-297
 keyboard, 23-33
 keyboard (DOS), 337-348
 keyboard (OS/2), 281-291
 make, 47-49
 mouse, 69-70, 73
 mouse (DOS), 375-377
 mouse (OS/2), 319-322

functions (*cont.*)

- naming conventions, 105
- prefix list, 4
- printer, 219-221
- printer (DOS), 387-390
- printer (OS/2), 331-334
- screen, 44-47
- screen (DOS), 355-374
- screen (OS/2), 299-317
- windows, 107-111
- windows (DOS), 379-386
- windows (OS/2), 323-330

I

interface, application programmer's (*see* Application Programmer's Interface)

K

- kb_char, 24, 26-28, 283, 339
- kb_edit, 25, 31-33, 283, 340
- kb_read, 24-26, 282-283, 339
- kb_scan, 24, 28-29, 283, 340
- kb_status, 24, 29-31, 282
- keyboards and keys, 23-33 (*see also* mouse systems)
 - alphanumeric data field entry dialog box, 119-129
 - DOS functions, 337-348
 - function prototypes, 23, 281, 337
 - not stopping program/retrieving 16-bit key code, 29-31
 - OS/2 full-screen functions, 281-291
 - retrieving character string, 31-33
 - stopping program execution/reading 16-bit key code, 25-26
 - stopping program execution/reading 8-bit character code, 26-27
 - stopping program execution/reading 8-bit scan code, 28-29
 - user interface becomes keyboard/mouse driven, 78-104

L

libraries

- building DOS, 19-20
- building OS/2, 18-21

M

- malloc, xiv
- memory, xiii-xiv
- menus (*see also* screen management; windows)
 - creating bar/drop-down windows, 52-68
 - dialog boxes (*see* dialog boxes)
 - Lotus grid style, 143-155
- mk_attr, 47, 49, 300, 356

- mk_attr_blink, 48, 301, 357
- mk_attr_intense, 47-48, 300, 356
- mk_attr_intense_blink, 48, 300, 356
- mk_attr_inverse, 48, 301, 357
- mk_char_attr, 49, 301, 357
- mk_token, 48-49, 301, 357
- mouse systems, 69-104
 - DOS functions, 375-377
 - function prototypes, 69, 319, 375
 - mapping dialog boxes, 111-119
 - mapping full-screen display, 71-73
 - mapping user interface, 73-78
 - OS/2 object module, 319-322
 - user interface becomes keyboard/mouse driven, 78-104
- ms_init, 69-70, 320, 376
- ms_map_display, 70, 73, 111, 228, 321-322, 377
- ms_off, 70, 320, 376
- ms_on, 70, 376
- ms_status, 70, 321, 377

O

- offRect, 305, 361
- operating systems (*see* disk operating system; OS/2)
- OS/2
 - advantages, 3
 - full-screen character mode API, 1-22
 - function prefix list, 4

P

- printers and printing, 219-278
 - command line utility, 221-227
 - DOS functions, 387-390
 - function prototypes, 219, 331, 387-390
 - OS/2 functions, 331-334
 - utility using multiple functions, 228-277
- print_char, 221, 333, 388
- print_close, 220, 334, 389
- print_cr, 220, 333, 389
- print_newline, 220, 333, 389
- print_open, 219-220, 333, 389
- print_set_column, 221, 334, 389-390
- print_string, 220-221, 332, 388
- programs (*see* source code)

R

- rdImg, 313, 372
- rdWind, 314, 373
- restRect, 52, 303, 359

S

- saveRect, 52, 302, 358
- screen management, 43-68 (*see also* menus; windows)

- color, 43
- DOS functions, 355-374
- function prototypes, 44, 299, 355
- mapping full-screen display using mouse, 71-73
- menu bar/drop-down window creation, 52-68
- OS/2 full-, 299-317
- useful routines, 49-51
- scrn_attr, 46, 52, 312, 365
- scrn_change_attr, 47, 49, 309, 367
- scrn_char, 45-46, 105, 110, 312, 364
- scrn_chr, 46, 309, 367
- scrn_clear, 45, 52, 309, 367
- scrn_init, 43-45, 49, 313, 364
- scrn_read_char, 47, 313, 368
- scrn_repeat_char, 46, 50, 310, 365
- scrn_restore, 45, 308, 371
- scrn_save, 45, 308, 371
- scrn_write, 45, 49, 105, 109, 311, 366
- setAttr, 327, 383
- setBord, 326, 382
- setRect, 52, 303, 359
- setTitle, 328, 383
- setWind, 325, 381
- sizeImg, 317
- sizeRect, 304, 360
- source code
 - ASCII.H, 12-13
 - BUILDLIB.BAT, 19-20
 - BUILDLIB.CMD, 18-19, 20-21
 - CURSOR.C, 294-297, 350-354
 - DFINST.C, 155-217
 - KB_STAT.ASM, 338
 - KEYBOARD.C, 282-291, 338-348
 - KEYBOARD.H, 8-12
 - MAKE.C, 300-302, 356-358
 - MOUSE.C, 319-322, 375-377
 - PRINTER.C, 332-334, 388-390
 - PROG2-1.C, 25-26
 - PROG2-2.C, 27-28
 - PROG2-3.C, 28-29
 - PROG2-4.C, 29-31
 - PROG2-5.C, 31-33
 - PROG3-1.C, 38-41
 - PROG4-1.C, 49-51
 - PROG4-2.C, 52-68
 - PROG5-1.C, 71-73
 - PROG5-2.C, 73-78
 - PROG5-3.C, 78-104
 - PROG6-1.C, 112-114
 - PROG6-2.C, 114-119
 - PROG6-3.C, 120-129
 - PROG6-4.C, 129-138
 - PROG6-5.C, 138-141
 - PROG6-6.C, 141-143
 - PROG6-7.C, 143-155
 - PROG6-8.C, 155-217
 - PROG7-1.C, 222-227
 - PROG7-2.C, 228-277
 - RECT.C, 302-307, 358-363
 - SCREEN.C, 307-317, 363-374
 - TPROTO.H, 4-8
 - TSTRUCT.H, 14-18
 - WINDOW.C, 324-330, 380-386
- strt_wind, 325, 381
- subRect, 304, 360

V

vdBox, 310

W

windows

- alphanumeric data field entry dialog box, 119-129
- border styles, 106
- character-based, 105-217
- commercial quality setup program, 155-217
- creating menu bars/drop-down, 52-68
- DOS functions, 379-386
- filename/file size dialog box, 141-143
- function prototypes, 379-380
- high-level function prototypes, 106-107, 323-324
- Lotus grid style menu, 143-155
- mapping dialog boxes, 111-119
- multiple item check dialog box, 129-138
- OS/2 full-screen functions, 323-330
- string list dialog box, 138-141
- wind_attr, 109, 327, 382
- wind_char, 105, 110, 330, 386
- wind_cu_move, 111
- wind_destroy, 111, 324, 380
- wind_display, 108
- wind_init, 107, 329, 384
- wind_kb_edit, 108, 329, 385
- wind_read_char, 111
- wind_remove, 108, 327, 383
- wind_repeat_char, 110, 330
- wind_write, 105, 109, 328, 384
- wrBox, 315
- wrImg, 314, 373
- wrWind, 315, 373-374

Other Bestsellers Of Related Interest

STACKER®:

An Illustrated Tutorial

—2nd Edition—Dan Gookin

Turn your single hard disk into two with this professional guide. Updated through Stacker 3.0, it contains information not found in the manuals. You'll use such features as Express or Custom Setup for Windows and DOS; Windows Stackometer™—a set of real-time gauges showing hard disk capacity, compression ratio, and fragmentation levels. Plus, you'll use Unstack™, a time-saving utility that decompresses files and automatically returns systems to their original state. 206 pages, 50 illustrations. Book No. 4447, \$19.95 paperback only

MICROSOFT ACCESS PROGRAMMING

—Namir C. Shammass

This hands-on introduction to Microsoft Access database programming is designed for anyone who's familiar with the BASIC language. It's a practical tutorial approach—complete with ready-to-use program code and professional tips, tricks, and warnings. You get up-to-date information on the built-in online help that Microsoft Access offers . . . how to craft the visual interface of a form . . . how to fine-tune the control settings to alter their appearance or behavior . . . and much more. 304 pages, 158 illustrations, 3.5" disk. Book No. 4333, \$32.95 paperback only

BUILD YOUR OWN 486/486SX AND SAVE A BUNDLE

—2nd Edition—Aubrey Pilgrim

This hands-on guide makes it possible for you to build your own state-of-the-art, 100% IBM-compatible PC for about one-third of the retail cost or less with little more than a few parts, a screwdriver, and a pair of pliers. So don't shell out huge sums of money for a PC at your local retail outlet. This book will allow you to enjoy the speed and power of a 486—and still put food on the table. 256 pages, 58 illustrations. Book No. 4270, \$29.95 hardcover only

WINDOWS® BITMAPPED GRAPHICS—Steve Rimmer

Stocked with ready-to-run source code in C, and illustrated with many fine examples of bit-mapped output, this complete programmer's reference gives you all the practical information you need to work effectively with Windows-compatible graphics formats, including Windows BMP, TIFF, PC Paintbrush, GEM/IMG, GIF, Targa, and MacPaint. You get a toolbox of portable source code designed to help you integrate these standards into your Windows applications plus a whole lot more. 400 pages, 82 illustrations. Book No. 4265, \$38.95 hardcover only

**BUILDER LITE:
Developing Dynamic Batch
Files—Ronny Richardson**

With this software and Richardson's accompanying user's manual, even beginners will be able to build and test sophisticated batch files in as little as 10 minutes. Richardson's step-by-step tutorial demonstrates how to write batch files that manipulate displays, create menus, make calculations, customize system files, and perform looping operations. This isn't a demo package, either. Builder Lite was developed by Doug Amaral of hyperkinetix, inc., especially for this book. 368 pages, 61 illustrations, 3.5" disk. Book No. 4248, \$44.95 hardcover only

**DR. BATCH FILE'S ULTIMATE
COLLECTION—Ronny
Richardson**

Boost productivity, enhance DOS performance, and save hundreds of unnecessary keystrokes with this practical library of programs—no programming skills required. Assembled here and on the FREE 3.5" companion disk are over 120 of the most useful batch files available for creating and using keyboard macros, saving and reusing command lines, tracking down viruses in COMMAND.COM, and much more. 440 pages, 146 illustrations, 3.5" disk. Book No. 4220, \$39.95 hardcover only

**NORTON DESKTOP® FOR
WINDOWS® 2.0:
An Illustrated Tutorial
—Richard Evans**

"Evans tells the reader virtually everything necessary to use the Norton Utilities . . . Recommended."

—Computer Shopper on a previous edition

This example-packed guide gives you step-by-step, illustrated instructions for using each Norton Desktop library—including valuable troubleshooting advice and solutions to common problems. Evans, whose previous books on the Norton Utilities have sold more than 50,000 copies, not only shows you how to optimize the Norton Desktop Utilities, he also demonstrates the use of Norton Disk Doctor and Norton Backup. 240 pages, 109 illustrations. Book No. 4208, \$29.95 hardcover only

**BUILD YOUR OWN COMPUTER
ACCESSORIES AND SAVE A
BUNDLE—Bonnie J. Hargrave
and Ted Dunning**

Here are step-by-step instructions for 27 useful network management and computer diagnostic devices. Practical guidance for building accessories make complex computer network operations easier and faster. Plus, you'll find a special section on the tools necessary for basic soldering and cabling operations, information on how to read circuit diagrams and schematics, a list of component suppliers, and estimated costs for each project. 376 pages, 222 illustrations. Book No. 4134, \$29.95 hardcover only

Prices Subject to Change Without Notice.

Look for These and Other TAB Books at Your Local Bookstore

To Order Call Toll Free 1-800-822-8158

(24-hour telephone service available.)

or write to TAB Books, Blue Ridge Summit, PA 17294-0840.

Title	Product No.	Quantity	Price

☐ Check or money order made payable to TAB Books

Charge my ☐ VISA ☐ MasterCard ☐ American Express

Acct. No. _____ Exp. _____

Signature: _____

Name: _____

Address: _____

City: _____

State: _____ Zip: _____

Subtotal \$ _____

Postage and Handling
(\$3.00 in U.S., \$5.00 outside U.S.) \$ _____

Add applicable state and local
sales tax \$ _____

TOTAL \$ _____

TAB Books catalog free with purchase; otherwise send \$1.00 in check or money order and receive \$1.00 credit on your next purchase.

Orders outside U.S. must pay with international money in U.S. dollars drawn on a U.S. bank.

TAB Guarantee: If for any reason you are not satisfied with the book(s) you order, simply return it (them) within 15 days and receive a full refund.

BC

Order Form for Readers Requiring a Single 5.25" Disk

This Windcrest/McGraw-Hill software product is also available on a 5.25"/1.2Mb disk. If you need the software in 5.25" format, simply follow these instructions:

- Complete the order form below. Be sure to include the exact title of the Windcrest/McGraw-Hill book for which you are requesting a replacement disk.
- Make check or money order made payable to *Glossbrenner's Choice*. The cost is \$5.00 (\$8.00 for shipments outside the U.S.) to cover media, postage, and handling. Pennsylvania residents, please add 6% sales tax.
- Foreign orders: please send an international money order or a check drawn on a bank with a U.S. clearing branch. We cannot accept foreign checks.
- Mail order form and payment to:

Glossbrenner's Choice
Attn: Windcrest/McGraw-Hill Disk Replacement
699 River Road
Yardley, PA 19067-1965

Your disks will be shipped via First Class Mail. Please allow one to two weeks for delivery.

.....✂️.....

Windcrest/McGraw-Hill Disk Replacement

Please send me a replacement disk in 5.25"/1.2Mb format for the following Windcrest/McGraw-Hill book:

Book Title _____

Name _____

Address _____

City/State/ZIP _____

DISK WARRANTY

This software is protected by both United States copyright law and international copyright treaty provision. You must treat this software just like a book, except that you may copy it into a computer in order to be used and you may make archival copies of the software for the sole purpose of backing up our software and protecting your investment from loss.

By saying “just like a book,” McGraw-Hill means, for example, that this software may be used by any number of people and may be freely moved from one computer location to another, so long as there is no possibility of its being used at one location or on one computer while it also is being used at another. Just as a book cannot be read by two different people in two different places at the same time, neither can the software be used by two different people in two different places at the same time (unless, of course, McGraw-Hill’s copyright is being violated).

LIMITED WARRANTY

Windcrest/McGraw-Hill takes great care to provide you with top-quality software, thoroughly checked to prevent virus infections. McGraw-Hill warrants the physical diskette(s) contained herein to be free of defects in materials and workmanship for a period of sixty days from the purchase date. If McGraw-Hill receives written notification within the warranty period of defects in materials or workmanship, and such notification is determined by McGraw-Hill to be correct, McGraw-Hill will replace the defective diskette(s). Send requests to:

Customer Service
Windcrest/McGraw-Hill
13311 Monterey Lane
Blue Ridge Summit, PA 17294-0850

The entire and exclusive liability and remedy for breach of this Limited Warranty shall be limited to replacement of defective diskette(s) and shall not include or extend to any claim for or right to cover any other damages, including but not limited to, loss of profit, data, or use of the software, or special, incidental, or consequential damages or other similar claims, even if McGraw-Hill has been specifically advised of the possibility of such damages. In no event will McGraw-Hill’s liability for any damages to you or any other person ever exceed the lower of suggested list price or actual price paid for the license to use the software, regardless of any form of the claim.

McGRAW-HILL, INC. SPECIFICALLY DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Specifically, McGraw-Hill makes no representation or warranty that the software is fit for any particular purpose and any implied warranty of merchantability is limited to the sixty-day duration of the Limited Warranty covering the physical diskette(s) only (and not the software) and is otherwise expressly and specifically disclaimed.

This limited warranty gives you specific legal rights; you may have others which may vary from state to state. Some states do not allow the exclusion of incidental or consequential damages, or the limitation on how long an implied warranty lasts, so some of the above may not apply to you.

If you need help with the enclosed disk . . .

The disk included with this book contains six executable zipfiles (BCOS2LIB, DEMOSRC, DOSLIB, ICOS2LIB, INCLUDE, and OS2EXE) and a README file. You might find it more convenient to put these files on your hard drive.

To create a subdirectory in your root C drive that you can keep these files in, type

```
MKDIR directory-name
```

at your C:\hard drive prompt, where *directory-name* is what you want to name the subdirectory (probably "INSTANT" or "DORFMAN" or "4472"—whatever is convenient). You could also create subdirectories for each one of the zipfiles on the disk, so that all the different files won't be jumbled together when you unzip them all.

To copy all the files to one subdirectory, place your disk in your floppy drive (probably drive B) and type

```
COPY B:\*.* C:\directory-name
```

The files on the disk will now be copied to your newly created subdirectory. To copy just one zipfile to a subdirectory, place your disk in your floppy drive (probably drive B) and type

```
COPY B:\zipfile-name C:\directory-name
```

where *zipfile-name* is the name of one specific zipfile on the disk.


To unzip any of the zipfiles, simply type its name (i.e., run it like an executable file) and hit Enter. For example, type

```
C:\directory-name\INCLUDE
```

to unzip the INCLUDE zipfile.

Important

Read the Disk Warranty terms on the previous page before opening the disk envelope. Opening the envelope constitutes acceptance of these terms and renders this entire book-disk package nonreturnable except for replacement in kind due to material defects.



Make your
DOS code run
instantly under
OS/2!



PROGRAMMING
Intermediate/Advanced

Windcrest®/McGraw-Hill
Blue Ridge Summit, PA 17294-0850

"We really enjoyed the concept and implementation of the similar DOS and OS/2 libraries for character-based programming. Excellent work."

—Brian Curran
OS/2 Application Development Specialist

If you're a programmer who wants to write character-mode DOS applications that run under full-screen OS/2, *Instant OS/2!* is THE book for you. With this time-saving guide, you can create programs and interfaces for DOS and OS/2 *using the same source code!*

Len Dorfman has already done the work—all you have to do is slip in the accompanying disk, follow the step-by-step instructions, and your old DOS routines become instantly portable to OS/2. Inside, you'll find more than 20 sample programs that demonstrate how to create:

- Mouse-driven character-based windows
- A menu bar that exceeds industry standards for drop-down window user interfaces
- Formatted printer output
- A mouse-based coordinate map of the screen
- Colorful screen displays
- And much more!

\$34.95

0993

ISBN 0-8306-4522-5

